175 PTAS
81
mi computer
CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR
WEDGE
EDGE
WEDGE
; WEDGE STARTS HERE
WEDGE
LDA COUNT
CMP #RATE
BEQ CONT
; DO CLOCK THIS?
NO
OUT
JMP EXIT
RESET IRQ COUNTER
CONT LDA #$FF
STA COUNT
LDA DISPLY
BEQ OUT
; NO...BRANCH
; GET HOURS/AM/PM
; PUT A COPY IN X REG
; GET AM/PM
; BRANCH IF PM
; DISPLAY 'A'
CLOCK+3
AND #$80
BNE PM
#AY
MERIDP
LDA #PEE
MERIDP
STA SCNLOC+10
LDA COLOR
STA COLLOC+10
GET
Editorial Delta, S.A.

# MI COMPUTER CURSO PRACTICO
## DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR


Publicado por Editorial Delta, S.A., Barcelona

Volumen VII-Fascículo 81

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Paseo de Gracia, 88, 5.°, 08008 Barcelona
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE


MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.



ISBN: 84-85822-83-8 (fascículo) 84-7598-067-2 (tomo 7)
84-85822-82-X (obra completa)
Depósito Legal: B. 52-84

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 318507
Impreso en España-Printed in Spain-Julio 1985


Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.


Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.


Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

**Servicio de suscripciones y atrasados** (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S.A. (Paseo de Gracia, 88, 5.°, 08008 Barcelona), o también con talón bancario remitido a la misma dirección.

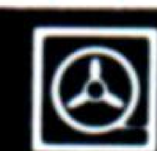
c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

**No se efectúan envíos contra reembolso.**





# Lenguajes modernos

**¿Cuáles son las causas de la controversia existente respecto a enseñar programación de ordenadores a alumnos de escuela primaria?**

El papel tradicional del niño en la educación es el de receptor del aprendizaje. El conferirle un papel activo suscita una controversia inevitable, por lo que no debe resultar sorprendente el que educar al niño en el campo de la programación sea una tarea de elevado riesgo tanto para los maestros como para los alumnos.

Los estudios de informática se incorporaron a los planes de estudio a finales de los años setenta y han ido adquiriendo una creciente aceptación entre los alumnos y los padres, indistintamente. Encuestas realizadas en Gran Bretaña entre los padres a comienzos de los ochenta revelaron que éstos consideraban los estudios de informática como la tercera asignatura escolar en escala de importancia (tras matemáticas e inglés); pero este entusiasmo no siempre es compartido por los miembros del cuerpo de profesores, ni tampoco por las personas pertenecientes a la industria del ordenador, a la caza de nuevos adeptos.

Algunos maestros, por ejemplo, piensan que los ordenadores deberían integrarse en todas las áreas de los planes de estudio en vez de otorgárseles un status separado, dominio exclusivo de un Departamento de Estudios Informáticos. Pero en algunas ocasiones, esto ha significado que la ciencia informática, como disciplina, se haya visto afectada por algunos problemas, surgidos, especialmente, a consecuencia de los escasos recursos económicos de que disponen la mayoría de las escuelas. A pesar de ello, muchos especialistas de la educación continúan pensando de forma muy apasionada en los beneficios que reporta la programación *per se*.

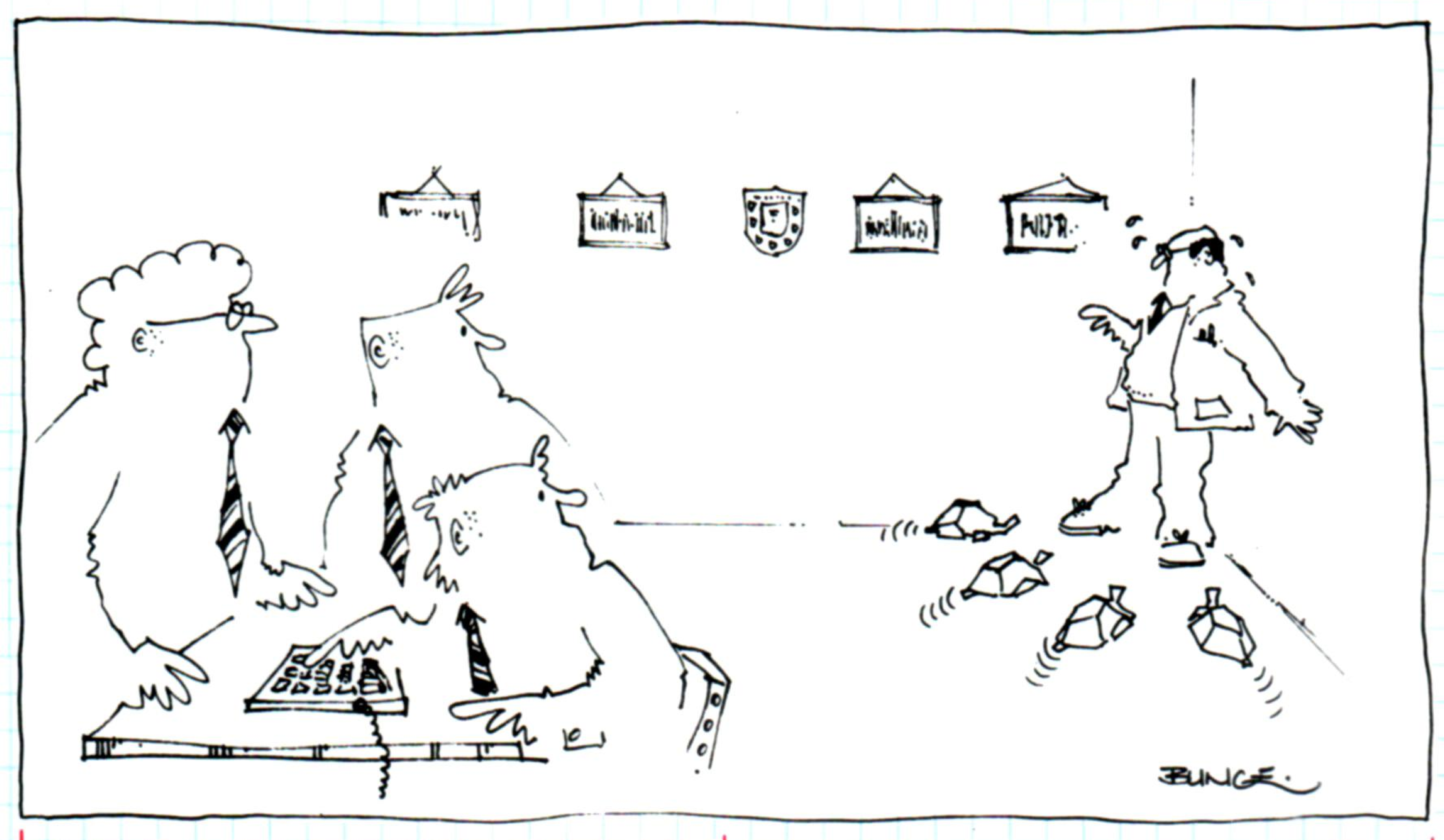
El hecho de que un niño no llegue jamás a convertirse en un artista o un escritor profesional, no justifica la omisión de estos estudios. Como ya hemos visto a lo largo de esta serie, a los niños pequeños se los ayuda a aprender conceptos y habilidades normalmente consideradas fuera de su comprensión como un derivado de la programación, y estos "beneficios colaterales" deben tener por sí mismos alguna justificación, al menos para enseñar programación en las escuelas.

Hasta hace poco tiempo, la enseñanza de programación en las escuelas se ha visto influida en gran medida por el hecho de que la mayoría de las máquinas se suministran con BASIC. Debido a su contenido abstracto, el BASIC está fuera del alcance de la mayoría de los niños menores de 12 años, de modo que en algunos casos el estudio de informática se ha limitado a las escuelas de nivel secundario (a partir de sexto curso).

**Opciones de lenguajes**
La implementación de diferentes lenguajes en los micros poco a poco está ayudando a ensanchar la base de los estudios de programación en las escuelas. Algunos lenguajes, como el PROLOG, se pueden estudiar provechosamente sobre la base de su propia estructura. Otros, como el COMAL y el LOGO, son valiosos porque ofrecen una introducción más disciplinada a la programación que el BASIC (que, aunque es fácil de aprender, también puede propiciar una utilización abusiva y llevar a técnicas de programación descuidadas)



Alan Adler



Además, a pesar de su utilidad en otros campos, la industria informática generalmente se muestra reticente respecto al empleo del BASIC en la educación, debido a su enfoque no estructurado y a la forma en que puede propiciar la adquisición de malos hábitos de programación en el principiante. Lamentablemente, muchos maestros del área de estudios informáticos se han introducido por propia iniciativa en el tema mediante cortos cursillos de BASIC, propagando de ese modo su influencia a pesar de la disponibilidad de otros lenguajes, como el LOGO, COMAL y PROLOG.

## Si, como se ha sugerido ampliamente, el empleo del BASIC favorece malas técnicas de programación y un pensamiento desordenado o abstracto, ¿cuáles son las alternativas?

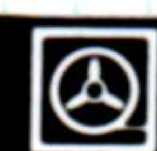
La introducción de versiones más estructuradas del BASIC (el BASIC BBC, p. ej.) ha ayudado a atenuar algunos recelos industriales, pero el verdadero interés por la programación educativa reside en los tres lenguajes que hemos mencionado anteriormente. A lo largo de esta serie ya hemos visto con cierto detalle el LOGO. Se está promoviendo de forma activa en Gran Bretaña y Estados Unidos a través del British LOGO Users Group y la Young Persons LOGO Association, respectivamente. Todos los fabricantes de micros populares han sacado a la venta versiones del lenguaje, rescatándolo de la oscuridad de los proyectos de investigación de inteligencia artificial, en los que en algún momento pareció destinado a quedar confinado.

En el State Teacher's College, de Tonder (Dinamarca), Borge Christensen desarrolló el COMAL (COMmon Algorithmic Language). El COMAL, según su creador, es una forma estructurada del BASIC. Christensen y sus colegas se encontraban con que los programas en BASIC de sus alumnos estaban tan mal escritos que eran casi incomprensibles. "Tras procesar cantidades ingentes de programas en BASIC —comentó—, comprendí que algo olía a podrido en Dinamarca." Los argumentos de Christensen en favor del COMAL han ejercido influencia en el gobierno de Dinamarca, al punto de que se ha adoptado el COMAL como primer lenguaje de programación en las escuelas danesas. También se lo prefiere, en detrimento del BASIC, en la República de Irlanda, y en Gran Bretaña ha despertado cierto interés. En una escuela secundaria del sur de Londres se inició un ambicioso proyecto en el cual los alumnos debían escribir un programa de localización de satélites en COMAL.

Un pequeño grupo de especialistas en educación de Inglaterra ha estado utilizando el PROLOG (PROgramming in LOGic). Empleando el PROLOG, un problema expresado en términos lógicos se puede traducir fácilmente a un programa para ordenador. En Park House School (Londres) se planificó un proyecto para desarrollar material para usar en la enseñanza de lógica como una disciplina de programación para alumnos de entre 10 y 13 años. Algunos de éstos aprendieron a programar fluidamente y otros, aunque participaron en los trabajos de la clase, escribieron poco por sí mismos. Con posterioridad, a medida que el conocimiento de la informática se generalizó en toda la escuela, en las clases posteriores se consiguió un mayor éxito. El proyecto le ha dado más importancia a la enseñanza de la lógica que a la de programación.

## ¿Cuáles son las diferencias existentes entre estos lenguajes que podrían determinar lo que aprenda el niño a partir de la programación?

El LOGO ha sido objeto de profundas investigaciones, en un intento por dar una respuesta a los críticos que afirman que existen pocas pruebas de los beneficios que obtienen los niños a través de la programación en este lenguaje. (Lamentablemente, la relativa minoría de escuelas que utilizan COMAL y PROLOG impide que estos len-



guajes se puedan estudiar adecuadamente.) En Estados Unidos, la National Science Foundation financió el proyecto de LOGO Brookline para estudiar el empleo del LOGO por parte de los niños. La primera fase del proyecto implicó poner a los alumnos de sexto curso (11 años de edad) en un laboratorio de ordenadores y observar sus características personales y sus actividades. En la segunda fase los ordenadores se introdujeron en la escuela y se trabajó parcialmente desarrollando material para los planes de estudio. Un resultado interesante de la primera fase fue que todos los niños realizaron con éxito su trabajo en LOGO, independientemente de la capacidad demostrada en otras asignaturas. Este mismo resultado se ha producido en otros proyectos de LOGO.

## ¿Qué otros intentos se han realizado para estudiar los efectos en niños que han aprendido a programar?

La Academia de Ciencias de Nueva York y tres escuelas de esta misma ciudad prepararon un proyecto para dar entrenamiento y apoyo a la enseñanza del LOGO en las escuelas. En todas las aulas se instalaron ordenadores, miembros del proyecto visitaron semanalmente las escuelas, y maestros voluntarios participaron en seminarios mensuales. El personal calculó que el éxito del proyecto alcanzó a casi el 95 % de las aulas. El director del proyecto señaló numerosas similitudes con el proyecto Brookline. Una consecuencia fundamental de disponer del LOGO en la clase fue la positiva interacción entre los estudiantes. Los niños que no habían obtenido resultados satisfactorios en otras asignaturas se desenvolvieron muy bien con el LOGO. "El beneficio educativo fue evidente —manifestó—. Los niños se comprometieron en procesos intelectuales y sociales válidos. Hemos observado cambios notables en las relaciones de los niños con la escuela y con el aprendizaje." Un producto del proyecto fue la recomendación de contar en la zona con una "escuela imán" que actuara como eje para las investigaciones centradas en el LOGO.

En el Center for Children and Technology de Nueva York se llevó a cabo un proyecto de 30 meses para evaluar la transferencia de estrategias de resolución de problemas a otras áreas de los planes de estudio, y para juzgar el efecto del trabajo orientado hacia el LOGO sobre la interacción social.

Es interesante observar la forma en que los ordenadores han estimulado la cooperación e interacción entre los niños. "Durante la actividad con el ordenador hubo más interacción en la tarea que durante cualquier otra actividad de la clase no dirigida por el maestro", observó el jefe del programa.

## ¿Qué problemas, si los hubiera, han obstaculizado la eficacia de estos estudios?

Son muchísimos los problemas que surgen al investigar la efectividad de la programación en las escuelas. Los pedagogos se muestran reacios a asignar mucho tiempo a la programación cuando los beneficios de su inclusión en los planes de estudio no parecen evidentes. Los fondos de financiación y, por consiguiente, el hardware son escasos. También es difícil llevar a cabo un estudio controlado, puesto que la creación de un grupo de control en la misma escuela implica privar a otros niños de horas frente al ordenador, mientras que tener un grupo de control en otra escuela dificulta la comparación, debido a los diferentes programas de estudio y métodos de enseñanza. Muchos maestros están tan poco familiarizados con los ordenadores como los propios niños, de modo que con frecuencia éstos no disponen de ninguna fuente de conocimientos a la cual recurrir.

## Aparte de desarrollar enfoques más lógicos, ¿qué otras aplicaciones prácticas puede esperar hallar el niño después de aprender a programar?

Una vez que el niño se ha introducido en el arte de la programación, la cantidad de aplicaciones potenciales es inmensa. Un grupo de niños ha utilizado tortugas para coreografía; habiendo programado éstas para desplazarse en formación, transformaron luego los movimientos de los dispositivos en danza. Las características del LOGO para la programación de palabras y listas se han empleado para generar historias en las clases de lengua y literatura. Se les entrega a los niños una serie de bifurcaciones que ilustran las diversas opciones que determinan la dirección del cuento. Además, el LOGO posee amplias facilidades de sonido. De la misma forma en que se utiliza CUADRADO para definir PATRON, NOTA puede definir ACORDE, ACORDE puede definir MELODIA, y así sucesivamente, permitiendo, por consiguiente, que los niños integren su uso del ordenador con sus estudios de música.

## Si son pocas las escuelas que enseñan programación, y en lenguajes diferentes, ¿cómo podrían extenderse los posibles beneficios?

En Gran Bretaña, al menos, el continuo establecimiento del LOGO como el lenguaje de programación para los niños pequeños ha supuesto la aparición de nuevas e interesantes áreas de actividad del plan de estudios. Pero al nivel secundario, el BASIC continúa disfrutando de su dominio y es necesario poner en práctica algunas medidas para asegurar que el continuo apoyo que se presta al LOGO se extienda a todo el sistema educativo. De lo contrario, la confusión que rodea al estudio de informática en las escuelas probablemente irá en aumento, cuando los programadores cualificados de LOGO vayan subiendo de nivel en el sistema y se encuentren con que han de abordar el BASIC. Tal como sucede con los ordenadores, su empleo en las escuelas también se debe estandarizar para que pueda producir beneficios a largo plazo.

Aprender a programar le ofrece al niño otro medio de expresión y creatividad y le estimula a pensar en una forma precisa y organizada.

# Digitando

## En nuestra serie sobre bases de datos vamos a examinar ahora las "claves" primarias y secundarias

Una *clave* es una herramienta incorporada en el software del administrador de bases de datos que ayuda al usuario a localizar un registro específico. Consideremos el manual de mantenimiento de un coche. Si usted desea ajustar el carburador, la información relativa a la forma de hacerlo estará oculta en algún sitio del manual. Podría estar, por ejemplo, en la página 36, de modo que este número de página se podría considerar como la clave para la información que necesita. Sin embargo, lo más probable es que no sepa que del carburador se habla en la página 36, de modo que lo que deberá hacer es remitirse al índice. Éste le proporcionará el número de página que necesita, permitiéndole acudir a la parte correcta del manual sin tener que ir pasando página por página.

Y lo mismo sucede con un DBM. Cada registro de un archivo de base de datos tiene un número de registro exclusivo (conocido por *clave primaria*), pero para hallar un registro específico usted tiene la alternativa de mirar cada registro en secuencia hasta hallar el buscado o, si ya sabe el número de registro, ir directamente al registro en cuestión. Asimismo, también tiene la alternativa de utilizar uno de los campos como clave (denominado técnicamente *clave secundaria*). Si tuviéramos una base de datos sobre servicio de coches, emplearíamos el campo NOMBRECOMPONENTE como clave.

La mayoría de los DBM permiten designar campos específicos como "campos clave". Cuando un campo (NOMBRECOMPONENTE, en este caso) se ha designado como un campo clave, el DBM mantiene una tabla interna de palabras (series de caracteres) del campo especificado, junto con el número de registro apropiado (clave primaria). Cuando usted busca el registro sobre el carburador, el DBM busca en la tabla NOMBRECOMPONENTE hasta encontrar la serie de caracteres "carburador", ve cuál es el número de registro adecuado y después extrae ese registro. He aquí cómo podría implementar tal esquema en un DBM simple en BASIC:

```
INPUT'ENTRE CAMPO CLAVE';CCLAVE$
INPUT'ENTRE PALABRA A BUSCAR';PAL$
GOSUB 20000:REM SUBRUTINA DE BUSQUEDA
PRINT REGRESULT
```

Todo cuanto está sucediendo aquí es que se ha seleccionado una de varias matrices, utilizando CCLAVE$ y la subrutina ha efectuado la búsqueda empleando la serie PAL$ como una clave de búsqueda. Si la subrutina de búsqueda es suficientemente potente, será capaz de admitir pequeños errores de digitación y aun así hallar un registro probable. Una rutina sencilla como ésta no necesita depender de que se mantenga una tabla de entradas clave: bastarán procedimientos normales de búsqueda a través de todos los registros.

En el pasado, los DBM que han operado en microordenadores han sido bastante simples, como el que podría escribir usted en BASIC en un tiempo moderadamente corto. No obstante, el advenimiento del Sinclair QL, con su procesador 68000, supuso una importantísima brecha. Psion Ltd, utilizando potentes miniordenadores VAX, desarrolló el *Archive* para el QL y puso al alcance del usuario de un micro personal capacidades avanzadas para administración de bases de datos. Para ver cómo trabaja *Archive*, vamos a crear una sencilla base de datos sobre mantenimiento de coches, con apenas cuatro registros. Cada uno se compondrá de sólo dos campos, NOMBRECOMPONENTE y SERVICIO:

```
Carburador
Quitar cubierta y girar perilla

Depósito de aceite
Levantar tapa y llenar con aceite

Batería
Abrir tapa y llenar con agua destilada

Radiador
Abrir tapa y llenar con agua
```

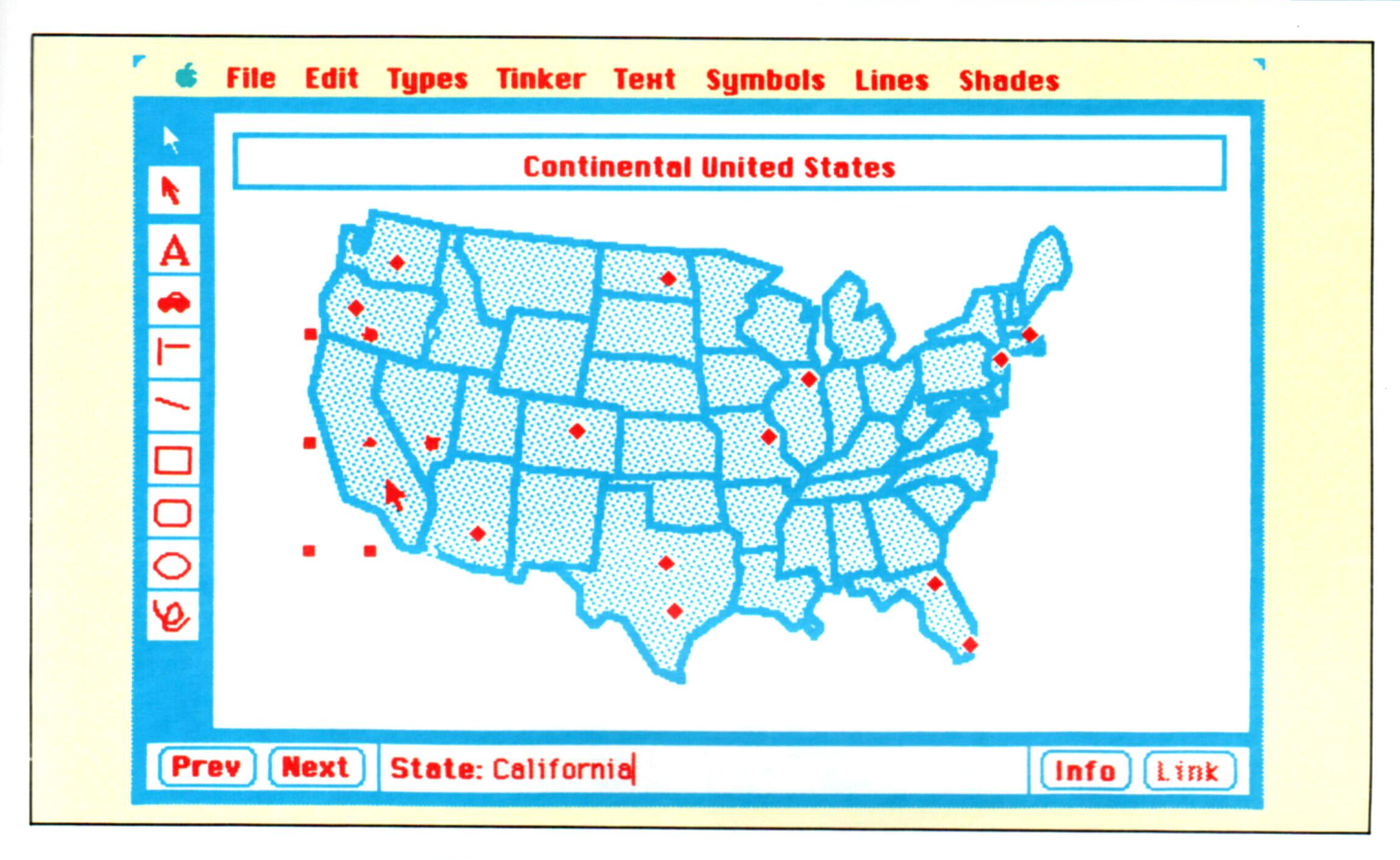
Cuando se ejecuta *Archive* en el QL, al usuario se le presenta una pantalla dividida en tres zonas: una de petición de instrucciones en la parte superior de la pantalla, otra de trabajo en el centro y una tercera de visualización en la parte inferior. Para comenzar un nuevo archivo de base de datos, seleccione la instrucción CREATE. Para crear nuestra base de datos de mantenimiento de automóviles, digite CREATE'COCHE' <CR> ('COCHE' es el nombre que le damos al archivo). Luego entre los nombres de los campos:

```
NOMBRECOMPONENTE$ <CR>
SERVICIO$ <CR>
<CR>
```

El signo $ añadido a los nombres de los campos indica que el campo está compuesto por una serie de caracteres. El <CR> final termina el proceso.

Para añadir registros a la base de datos, se digita la instrucción INSERT<CR>. Ésta visualiza los nombres de campos en la zona de trabajo y permite entrar los datos para cada campo. La pulsación de F5 después de haber entrado correctamente un registro hace que éste se añada al archivo de la base de datos. Una vez digitados todos los registros que desea entrar, puede salir de la modalidad INSERT pulsando la tecla Escape. Se puede entonces salir del archivo digitando la instrucción CLOSE.

Para buscar un archivo *Archive* para un registro, primero debe ser abierto. Esto se puede hacer digitando ya sea la instrucción LOOK, que sólo permite examinar los registros, o bien OPEN, que permite buscar registros y modificarlos. Habiendo abierto un archivo con OPEN o con LOOK, usted puede




entrar instrucciones simples tales como FIRST (para ver el primer registro del archivo), LAST (para ver el último registro), NEXT (para ver el registro siguiente) o BACK (para ver el registro anterior).

La búsqueda de un registro implica añadirle a la instrucción un argumento, como en FIND 'CARBURADOR'. Esta instrucción buscará todos los campos hasta localizar una entrada que concuerde con la parte entre comillas. Asimismo, se pueden utilizar operadores lógicos, como en SEARCH NOMBRECOMPONENTES$='CARBURADOR' AND SERVICIO$='GIRAR'. Esta instrucción buscará en el archivo hasta hallar un registro que contenga la serie CARBURADOR en el campo NOMBRECOMPONENTES$ *y* una serie GIRAR en el campo SERVICIO$ del mismo registro. El operador OR se puede emplear de la misma manera, de modo que se localizará un registro si se puede emparejar la serie especificada para un campo *o* (OR) la serie especificada para el otro campo.

Los registros generalmente se entran en una base de datos de forma caprichosa, pero con frecuencia es necesario acceder a ellos o imprimirlos de acuerdo a cierto orden. Supongamos, por ejemplo, que se le ha encargado la tarea de entrar en una base de datos todos los libros de una biblioteca por autor, título, editorial e ISBN (International Standard Book Number: número de libro según estándar internacional). La forma más sencilla de hacer esto sería ir trabajando a lo largo de los estantes, entrando los datos para cada libro a medida que éstos le van llegando a las manos. Utilizando las capacidades para clasificación que tenga incorporadas su DBM, podría luego clasificar los registros por orden alfabético (por autor, título, etc.), o incluso numéricamente según el ISBN.

Se puede, además, combinar una clasificación primaria con una clasificación secundaria. Supongamos que usted quisiera imprimir el archivo por orden alfabético de autor, con las entradas para cada autor dispuestas por orden alfabético de títulos. Ello permitiría salidas impresas como ésta:

```
CORTAZAR, JULIO
   Octaedro
   Alianza Editorial, S. A.
   84-206-3010-1
CORTAZAR, JULIO
   Queremos tanto a Glenda
   Alfaguara, S. A.
   84-204-2126-X
CORTAZAR, JULIO
   Rayuela
   Bruguera, S. A.
   84-020-6740-9
CORTES, HERNAN
   Cartas de relación de la conquista de México
   Espasa-Calpe, S. A.
   84-239-0547-0
```

Si los nombres de campos del *Archive* fueran AUTOR$, TITULO$, EDITORIAL$ e ISBN, clasificar los registros primero por autor y luego por título sería tan simple como:

```
ORDER AUTOR$;A,TITULO$;A
```

En este ejemplo, la A significa "listar por orden alfabético ascendente".

A estas alturas, ya resulta evidente que los DBM pueden convertir en un trabajo sencillo tareas que consumen tanto tiempo, como son la localización de registros y su reordenamiento de formas específicas. Próximamente analizaremos algunas de las facilidades más avanzadas que tienen incorporadas muchos administradores de bases de datos.

**Estados Unidos al dedillo**
*Filevision* se desarrolló para los Apple II y III, pero tuvo que esperar a que el mayor poder de proceso del Macintosh permitiera apreciar sus cualidades. Utiliza la interface para ratón del Macintosh para acceder a la base de datos "señalando y apretando". Al "frente" del programa hay un sofisticado paquete de dibujo que le permite crear una estructura de gráficos para su base de datos. La forma y los símbolos que usted dibuje se conectan a los campos de la base de datos. En la ilustración vemos la pantalla principal para una base de datos que contiene información relativa a Estados Unidos. Si señalamos el ratón para indicar California, un simple clic proporciona una breve cantidad de datos sobre ese estado; un doble clic abre un campo de texto que puede almacenar una cantidad de información muchísimo mayor. Los datos almacenados se pueden enlazar y clasificar de la misma forma que en cualquier base de datos normal. Por ejemplo, escogiendo "Highlight Some" del menú "Tinker", el programa le permitirá especificar un criterio de búsqueda y después visualizar los resultados dibujando con un trazo más grueso los contornos de esos estados en el mapa. Esta información también se puede imprimir seleccionando "Print" después de entrar el criterio especificado

# Para asir

**Llegados a este punto, nos corresponde diseñar el mecanismo de prensión y las conexiones necesarias para la "musculatura" del robot**

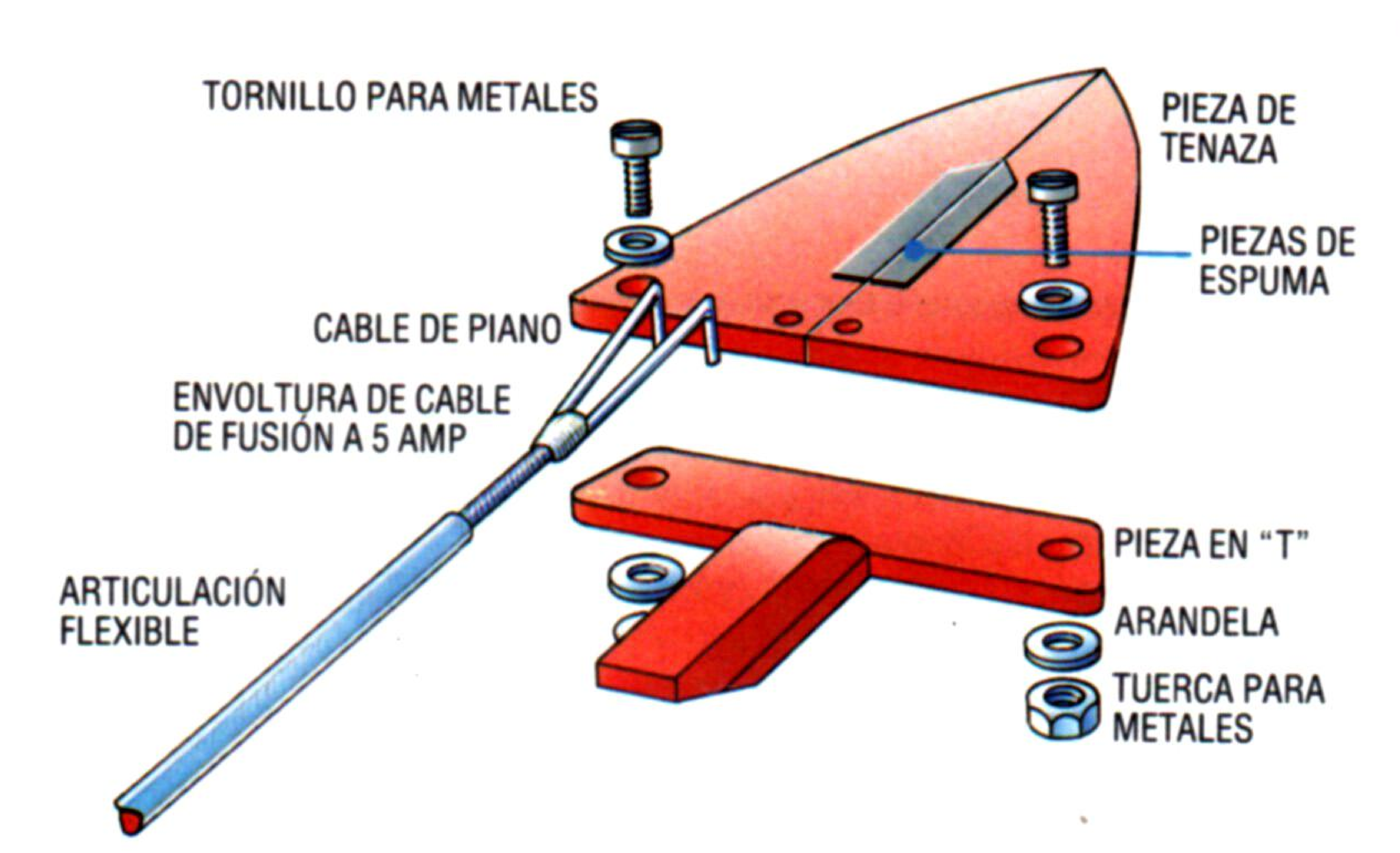
### Paso 1: Ensamblaje de la pinza

Taladre agujeros en los centros señalados en la pieza en "T" y las piezas de tenaza para poder colocar los tornillos. Pegue tiras de espuma en la quijada de las piezas de tenaza a modo de relleno. Se deben perforar otros dos agujeros en las posiciones señaladas en los cortes del patrón, de modo que por el agujero quepa estrechamente un trozo de cable de piano rígido. Las piezas de tenaza se pueden ajustar a la pieza en "T" mediante los tornillos y las tuercas para metal. Inserte arandelas arriba y abajo de las piezas de tenaza de modo que cuando se ajusten los tornillos las piezas de tenaza queden libres para pivotar alrededor del tornillo. Tomando un extremo de la articulación flexible de 500 mm, fije dos trozos de 30 mm de cable de piano al cable central envolviendo los extremos del cable de piano y el cable central con alambre de fusión a 5 amp y sellándolo con soldadura. Incline hacia abajo los otros extremos de los cables de piano, formando ángulos rectos, y empújelos en los agujeros preparados en las piezas de tenaza. El cable central y los cables de piano deben formar una "Y". Con las quijadas de la pinza cerradas, ajuste la carcasa de la articulación con algodón y péguela en su sitio.

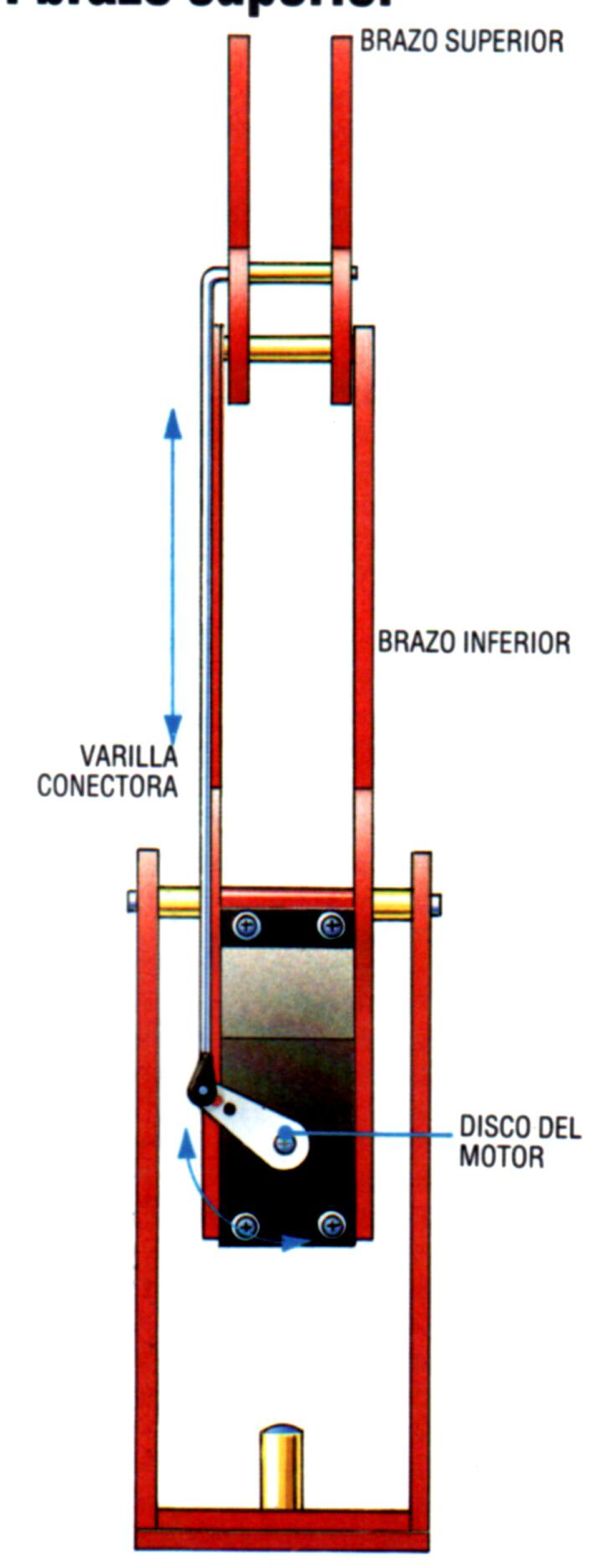
### Paso 2: Varilla conectora del brazo sup.

El brazo superior es controlado por el motor ya colocado en el ensamblaje del brazo inferior. El movimiento de rotación del husillo del motor se ha de convertir en un movimiento de tirar-empujar en el brazo superior. Ello se consigue ajustando un corto brazo de plástico (o "bocina") o un disco plástico de 30 mm al husillo. Ambos poseen un agujero por el cual se introducirá la varilla conectora. Aquí asumiremos que se están utilizando discos, pero el sistema funcionará perfectamente si en su lugar se emplean bocinas. Coloque el disco de 30 mm o la bocina en el husillo de modo que, cuando el motor

## Paso 1: Ensamblaje de la pinza

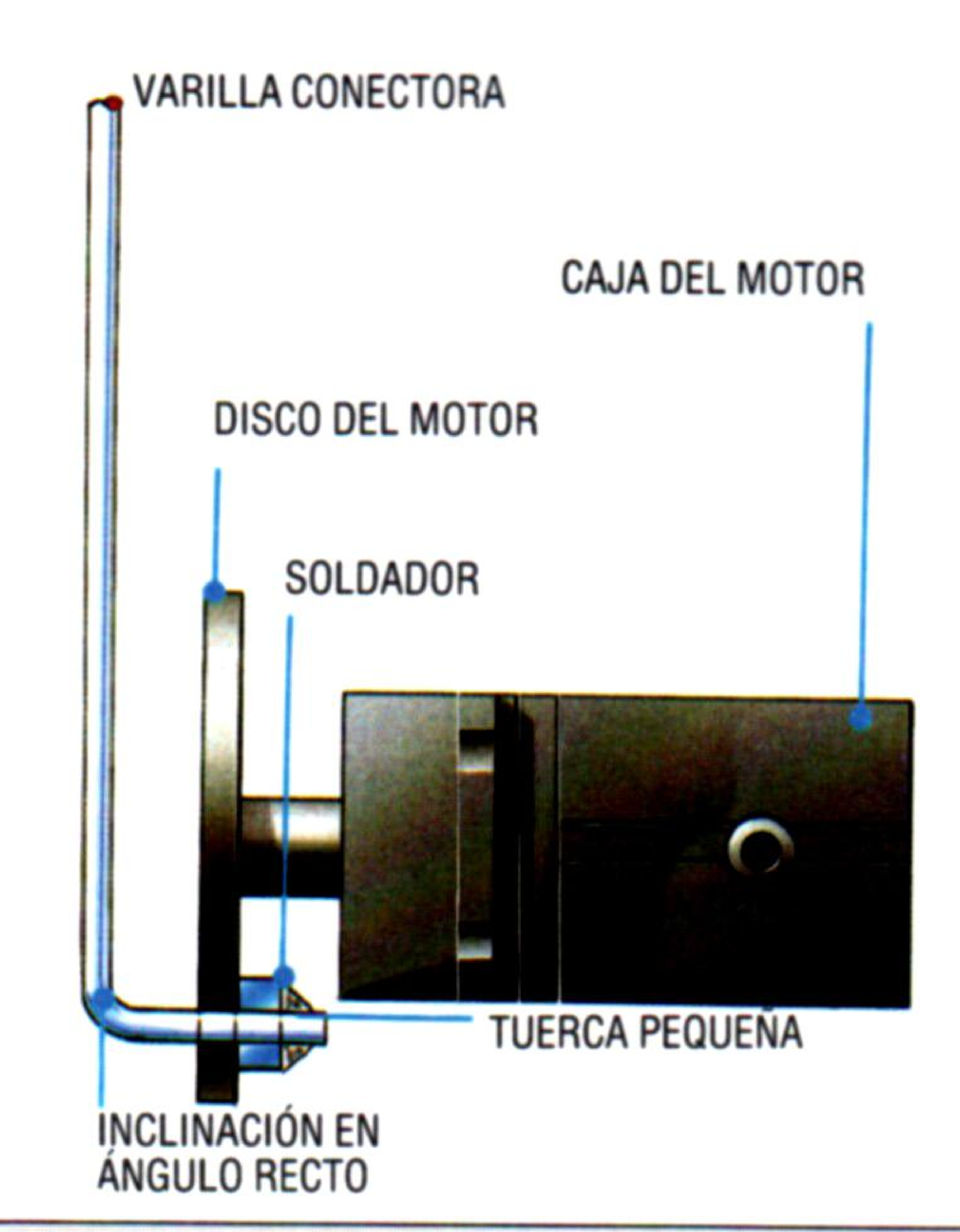



## Paso 2: Varilla conectora del brazo superior




## Conexión de las varillas a los discos

En el lugar donde adquiera los servomotores podrá también comprar varillas conectoras especiales. Si no las consigue, la mejor forma de conectar una varilla de acero de 2 mm normal al disco del motor consiste en efectuar una inclinación en ángulo recto cerca del extremo de la varilla, pasándola a través del agujero del disco y soldando una pequeña tuerca en la parte posterior del disco. Esto se puede realizar con más facilidad con el disco fuera del husillo del motor.
Donde las varillas se colocan en agujeros de los miembros de madera del brazo, es una buena idea alinear el interior del agujero con una delgada pieza de tubo de cobre de un diámetro interno apenas más grande que el de la varilla conectora. Ello reducirá la abrasión de la madera cuando el brazo se mueva






Kevin Jones



gire completamente en sentido antihorario, el agujero esté justo a la izquierda de la posición de las seis en punto (imagínese que el disco del motor es un cuadrante con las doce en punto situadas verticalmente hacia arriba). Tome un trozo de varilla de acero de 2 mm de diámetro e incline los extremos formando ángulo recto de modo que la varilla quepa a través del agujero del disco del motor o la bocina y los dos agujeros preparados para ella en las piezas del brazo superior, cerca de la juntura del codo. La longitud de la varilla debe ser tal que los brazos superior e inferior formen un ángulo de 90° en el codo cuando el motor gire completamente en sentido antihorario. Compruebe la acción del brazo girando lentamente el disco del motor, a mano, en sentido horario. Pegue o suelde la varilla en su sitio en el disco.

### Paso 3: Varilla conectora del brazo inferior

Monte los motores restantes en el cuerpo principal, como indica la ilustración. El motor de la derecha controla el movimiento del brazo inferior. Gire el motor de manera que quede por completo en sentido antihorario (visto desde el lado del disco del motor) y posicione el disco del motor de modo tal que el agujero de la circunferencia del disco quede justo a la derecha de la posición de las seis en punto. Ajuste un trozo de varilla de acero de 2 mm entre el disco del motor y los agujeros del brazo inferior que se hallan cerca de la juntura del hombro, como antes. La longitud de la varilla debe ser tal que el brazo inferior quede aproximadamente horizontal con el motor girado totalmente en contra de las agujas del reloj. Verifique la acción del brazo inferior haciendo girar a mano el disco del motor y pegue o suelde la varilla conectora al disco del motor.

### Paso 4: Conexión de la artic. al motor

El motor montado a la izquierda del cuerpo principal se utiliza para abrir y cerrar la pinza, empujando o tirando el cable central de la articulación flexible. Comience por taladrar un agujero a través de la pieza cruzada del cuerpo principal de modo que, como se ve en la ilustración, la articulación flexible se pueda deslizar a través del mismo desde atrás. Pegue la carcasa exterior de la articulación en este agujero. Cuando la cola esté seca, pase el cable central por este agujero, de modo que las quijadas de la pinza queden bien cerradas, y ajuste el extremo del cable al disco del motor. El agujero de la circunferencia del disco debe estar en las "nueve en punto" cuando el motor gire totalmente en sentido antihorario. El extremo del cable se puede fijar al disco del motor de varias maneras. Lo mejor es montar en el disco un pequeño prensacables. De lo contrario, se puede realizar una inclinación de 90° en el extremo del cable, y pegar o soldar en su sitio el extremo que se ha pasado a través del agujero del disco. Independientemente de cómo efectúe esta conexión, lo importante es que el motor pueda empujar y tirar del cable mientras vaya girando.

### Paso 5: Varilla del cuerpo principal

La última conexión a realizar es la del motor montado en la caja base al ensamblaje del cuerpo principal. Taladre un pequeño agujero en la base del cuerpo principal (necesitará correr el ensamblaje del tarugo de acero). El agujero se debe situar en línea con el pivote, junto al borde derecho (visto desde arriba) de la base del cuerpo principal. Realizado el agujero, vuelva a colocar el ensamblaje del cuerpo principal en el tarugo y gire hasta la posición de las dos en punto. Gire el motor por completo en sentido antihorario y posicione el disco de modo que el agujero de la circunferencia quede justo a la derecha de las seis en punto. Coloque una varilla de conexión como antes y compruebe que el cuerpo principal quede libre para girar en sentido antihorario.

## Paso 3: Varilla conectora del brazo inferior



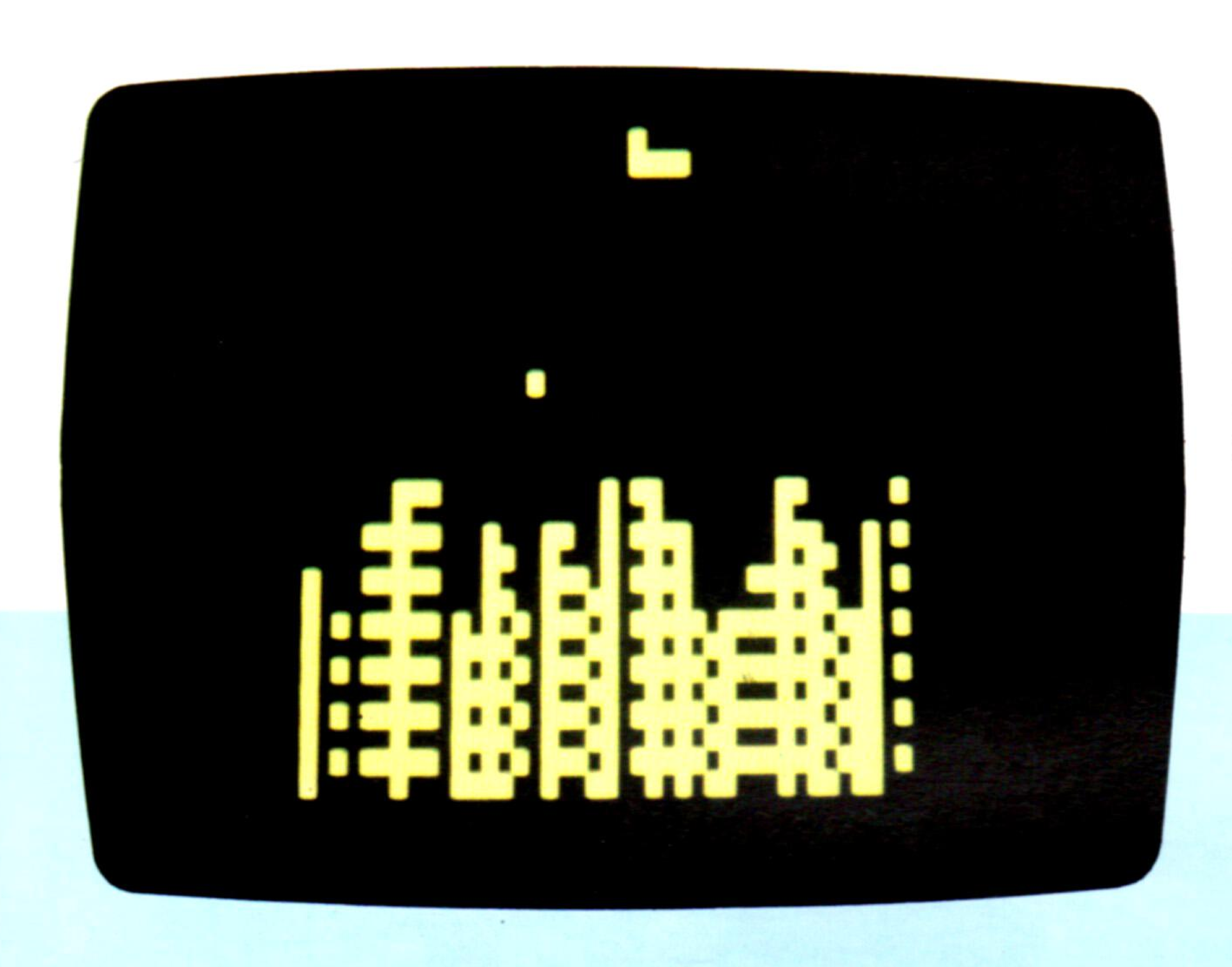
## Paso 4: Conexión de la articulación al motor



## Paso 5: Varilla del cuerpo principal

# Bombardeo aéreo

**La mayoría de los microordenadores permiten jugar al "bombardeo aéreo" u otros juegos parecidos que se conocen bajo otra denominación. El listado que ofrecemos es para el Dragon**



Su misión consiste en destruir la ciudad que está sobrevolando, con objeto de poder aterrizar. A cada pasada, su avión vuela un poco más bajo. No puede lanzar una bomba (pulsando sobre una tecla cualquiera) hasta que la bomba precedente haya alcanzado su objetivo o el suelo. Cuando su avión ha aterrizado (o cuando se ha estrellado contra un edificio), se registra la puntuación, así como el récord del día. Si este juego le parece demasiado difícil, puede variar los límites de la ciudad (6 y 26, línea 100).

```
10 REM ******************************
20 REM *     BOMBARDEO AEREO         *
30 REM ******************************
34 REM
35 REM INICIALIZACION
36 REM
39 REM A$=AVION
40 A$=CHR$(128)+CHR$(155)+CHR$(147)
49 REM B$=BOMBA
50 B$=CHR$(145)
58 REM H=POSICION DEL AVION Y
59 REM PUNTUACION
60 H=0
69 REM B=POSICION DE LA BOMBA
70 B=0
80 B1=B
90 CLS 0
94 REM
95 REM COLOCACION DE LA CIUDAD
96 REM
99 REM 6 y 26: LIMITES DE LA CIUDAD
100 FOR I=6 TO 26
110 C=RND (7)+151
114 REM LINEA 120:CAMBIAR 8
115 REM POR UN VALOR SUPERIOR
116 REM PARA DISMINUIR LA ALTURA
117 REM DE LAS CASAS
120 FOR J=15 TO RND(4)+8 STEP-1
130 PRINT@J*32+I,CHR$(C);
140 NEXT J
150 NEXT I
154 REM
155 REM BUCLE PRINCIPAL
156 REM
159 REM COLOCACION DEL AVION
160 PRINT@H,A$;
169 REM AVION ESTRELLADO?
170 IF PEEK(1027+H)<>128 THEN 240
179 REM LANZAMIENTO
180 IF INKEY$<>"" AND B=0 THEN B=H+33
189 REM BOMBA QUE LLEGA AL SUELO
190 IF B<>0 THEN GOSUB 360
199 REM NO SE HA SOLTADO LA BOMBA?
200 IF B=0 THEN GOSUB 400
209 REM AVANCE DEL AVION
210 H=H+1
219 REM AVION ATERRIZADO
220 IF H=507 THEN 240
230 GOTO 160
234 REM
235 REM AVION ATERRIZADO O ESTRELLADO?
236 REM
239 REM SE HA BATIDO EL RECORD?
240 IF H>R THEN R=H
250 PRINT@3,"PUNTUACION :";H,
260 PRINT@35,"RECORD :";R;
270 FOR I=1 TO 100
280 NEXT I
290 R$=INKEY$
300 PRINT@73,"OTRA (S/N) ?";
310 R$=INKEY$
320 IF R$="" THEN 310
330 IF R$<>"N" THEN 40
340 CLS
350 END
354 REM
355 REM BOMBA LANZADA
356 REM
359 REM HA LLEGADO AL SUELO
360 IF B>=510 THEN B=0
365 REM COLOCACION DE LA BOMBA
370 PRINT@B1,CHR$(128);
380 IF B<>0 THEN
    PRINT@B.B$;:B1=B:B=B+32
390 RETURN
394 REM
395 REM RETRASO PARA DISMINUIR
396 REM LA VELOCIDAD DEL AVION
397 REM SI NO SE HA LANZADO BOMBA
398 REM
400 FOR I=1 TO 20
420 RETURN
```

# Porción de memoria

## El Wafadrive Rotronics constituye una interesante alternativa de almacenamiento masivo para el Sinclair Spectrum

A pesar de que el Sinclair Spectrum se ha convertido en uno de los microordenadores personales más populares, ha sido objeto de muchas críticas en el sentido de que es una máquina inadecuada para el usuario "serio". Ello ha llevado a que se le considere sólo como una máquina para juegos. Parte del problema se ha centrado en el teclado, que no ha permitido que el usuario considere seriamente utilizar la máquina para aplicaciones tales como proceso de textos y administración de bases de datos. Recientemente, Sinclair Research ha intentado acallar algunas de estas críticas mediante el lanzamiento del Spectrum+, que viene equipado con un teclado estilo QL.

Sin embargo, esto no es más que una parte del problema. Otras de las dificultades que acosan a los que consideran al Spectrum como una máquina seria, son la falta de interfaces estándares del ordenador y, lo que es aún más importante, de un sistema de almacenamiento masivo fiable y rápido, algo vital para cualquier aplicación seria o de gestión que desee el aficionado. Por supuesto, con la introducción de la Interface 1 y el microdrive, el Spectrum estuvo al menos en condiciones de hacer disponibles estas facilidades. Pero los recelos no se disiparon, pues se consideró que el microdrive era lento y poco fiable. Además, si bien la base de software en cassette para el Spectrum es inmensa, es muy poco el software que se ha adaptado al medio que utiliza el microdrive. En una situación como ésta, los proveedores independientes tienden a cubrir el vacío y producir alternativas. Aquí vamos a analizar al primero de dos competidores en la carrera por dominar el mercado de almacenamiento masivo para el Spectrum: el Wafadrive de Rotronics. En el próximo capítulo nos concentraremos en el Discovery 1 de Opus Supplies.

**Espacio posterior**
A diferencia del propio sistema de almacenamiento masivo de Sinclair para el Spectrum, el Wafadrive de Rotronics viene en una única caja que contiene 2 unidades gemelas, una puerta RS232 y una interface Centronics. El cable plano se inserta en el conector marginal del Spectrum, con lo que las unidades se colocan convenientemente detrás del teclado, para un fácil acceso



Chris Stevens

A diferencia de la Interface 1 y los microdrives que lo acompañan, el Wafadrive de Rotronics es una unidad de "todo en uno". Ello significa que tanto las interfaces para periféricos como las unidades para almacenamiento masivo se hallan en una única caja. La ventaja de este sistema reside en que las unidades no poseen los cables de ampliación necesarios para el sistema Sinclair; no obstante, carecen de parte de la flexibilidad de los microdrives, que se pueden encadenar fácilmente en margarita para ampliar el espacio de almacenamiento.

## El aspecto de la máquina

El Wafadrive se halla alojado en una carcasa de plástico negro, con un cable plano de 35 vías que termina en una ranura para cartuchos instalada en el bus de ampliación del Spectrum. En la parte frontal de la unidad hay un par de ranuras para activar cartuchos *wafer*. Entre las ranuras hay tres diodos emisores de luz (LED). La luz del centro es el indicador de potencia, mientras que las otras dos indican la actividad de las unidades de disco.

En la parte posterior de la unidad hay tres conectores marginales. A la izquierda hay un bus de ampliación en paralelo que permite la conexión de la Interface 2. El conector marginal del centro es una interface compatible con Centronics que permite acoplarle a la unidad una impresora en paralelo. El tercero es una puerta en serie RS232 que permite la conexión en interface del dispositivo con modems y otros dispositivos en serie. Estas interfaces constituyen una mejora respecto a las que proporciona la Interface 1 de Sinclair, en la cual, por ejemplo, es necesario conectar una segunda interface Centronics al conector marginal de la unidad para poder operar impresoras en paralelo. Lamentablemente, sin embargo, los usuarios todavía tendrán que ir de tienda en tienda para conseguir impresoras Centronics o modems con conectores para cartuchos que sean compatibles con Wafadrive.

Los *wafers* (galletas) flexibles diseñados específicamente para el Wafadrive son en muchos sentidos similares a los utilizados en sus equivalentes de Sinclair. Dentro de cada *wafer* hay un bucle continuo de cinta de cassette tipo video, cuya anchura es de 1,8 mm. Se emplea esta cinta en vez de la de audio convencional debido a su mayor resistencia y capacidad de almacenamiento de información. Una vez formateada, esta cinta puede contener aproximadamente 128 K de datos, si bien Rotronics ha puesto a la venta también cartuchos de 64 y 16 K.

Los cartuchos son aproximadamente el doble de

anchos que los de Sinclair, aunque sus cajas protectoras tienen un largo y ancho similar. Ello les confiere a los cartuchos Rotronics una apariencia de cassettes en miniatura. Los cartuchos Wafadrive no necesitan carcasas protectoras, dado que la delicada cinta está protegida por una cubierta deslizante automática, de diseño similar a la de los microdiscos flexibles Sony de 3 ½ pulgadas, si bien la protección Rotronics es de plástico en vez de metal. En el lado izquierdo del *wafer* hay una traba de protección de escritura, que se puede quitar. Por supuesto, si esta traba se rompe ya no se podrá reemplazar, por lo que los usuarios habrán de pensar en otro método para poder utilizar sus cartuchos.

Las instrucciones que utiliza el Wafadrive son similares a las que emplean los microdrives de Sinclair. En ambos sistemas, la instrucción va seguida de un *, que indica que se ha de acceder al dispositivo de almacenamiento externo. Ejemplos de esta forma de empleo son SAVE*, LOAD* y VERIFY*. No obstante, el sistema Wafadrive posee ligeras diferencias, porque siempre hay presentes dos Wafadrives, frente a los numerosos microdrives que podrían incluirse en el sistema. Por ejemplo, cuando se formatea el microdrive uno de Sinclair, se utiliza la instrucción FORMAT 'm';0;"nombre", donde 'm';0 alude al número del microdrive que se está empleando. Cuando se utiliza el Wafadrive, la instrucción es FORMAT*'a:nombre', aludiendo la a: al nombre de la unidad de disco en uso. Observe que con el Wafadrive sólo puede haber una sección a: o b:, mientras que en el microdrive el número puede ser del cero al siete.

## El sistema de "corrientes"

El Wafadrive también saca provecho del sistema de "corrientes" (*streams*) utilizado en el Spectrum, en el cual hay 16 corrientes reservadas para el tratamiento de E/S; algunas de éstas están reservadas para uso de la pantalla y la impresora. Sin embargo, los canales del cuatro al quince están disponibles para otros dispositivos periféricos, y a las corrientes de salida al Wafadrive se accede mediante el empleo de la instrucción OPEN#. Asimismo, el Wafadrive añade al sistema dos corriente adicionales. Los canales *r* y *c* (estas letras también pueden ir en mayúscula) están reservados para las interfaces RS232 y Centronics, respectivamente, y su empleo es similar al de los canales *t* y *b* (utilizados cuando se accede a la puerta RS232 en la Interface 1).

A bordo hay una ROM de 8 K que contiene las instrucciones del BASIC extendido que se utilizan para controlar el sistema. Este sistema operativo Wafadrive (WOS: Wafadrive Operating System) es capaz de funcionar paginando los 8 K inferiores de la ROM del Spectrum, de forma muy similar a la forma en que lo hace la propia Interface 1. Por ejemplo, la instrucción LOAD* en realidad genera un error en el Spectrum; por consiguiente, cuando el intérprete de BASIC encuentre esta instrucción en la pantalla, llamará a la rutina para tratamiento de errores. No obstante, el WOS puede interceptar esta instrucción de llamada, paginándola entonces a la ROM del Wafadrive. Ésta, a su vez, asumirá el tratamiento de errores e interpretará LOAD* como una instrucción.

En comparación con los microdrives, el Wafadrive de Rotronics es más bien lento. Por ejemplo, un microdrive de 100 K requiere una media de 3 ½ segundos para localizar un elemento de información, que se transfiere luego al ordenador a una velocidad de hasta 19,2 Kbaudios. El Wafadrive, por su parte, sólo puede administrar una velocidad máxima de transferencia de 18 Kbaudios, con un tiempo máximo de acceso de 45 segundos en un *wafer* de 128 K. Ello representa una lentitud significativa, aunque esta relativa morosidad se compensa parcialmente gracias a su mayor fiabilidad. No obstante, se debe notar que los microdrives Sinclair fueron más rápidos que el Wafadrive en las mediciones realizadas en pruebas de comprobación.

Si bien estos tiempos de acceso son mucho más rápidos que los que posiblemente se consiguieran en cassette, aun así son más lentos que los tiempos comparables de unidades de disco. No obstante, el Wafadrive, al igual que los microdrives, incorpora un procedimiento práctico cuando se accede al catálogo de la cinta. El catálogo está retenido en el primer sector del cartucho tras el empalme que une entre sí los dos extremos de la cinta. Por consiguiente, para comprobar el CATálogo de un *wafer*, la unidad debe rebobinar la cinta hasta descubrir el empalme y poder leer el siguiente sector. Al cabo de varios segundos, el cabezal de la cinta habrá pasado sobre el sector del catálogo. Sin embargo, si se vuelve a entrar la instrucción CAT, en vez de rebobinar otra vez toda la cinta la unidad vuelve a visualizar otra vez el catálogo tras una fracción de segundo: el mismo queda retenido en RAM una vez que

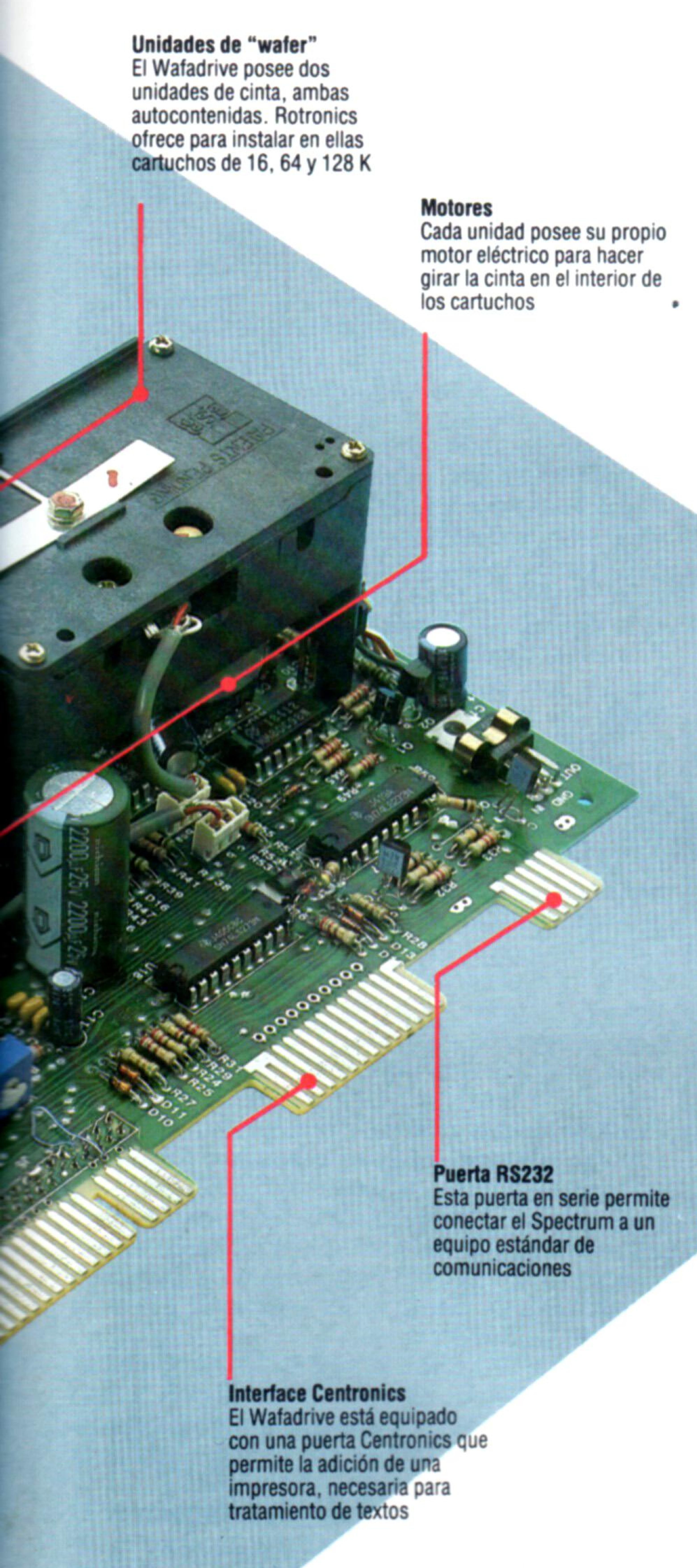
**Luces de estado**
Estos LED le dicen al usuario cuál es la unidad a la cual se está accediendo en cada momento

**Cable de conexión**
El cable plano de 54 vías se conecta al propio conector marginal del Spectrum. El Wafadrive no sólo recibe sus datos controladores a través de este bus, sino también su fuente de alimentación eléctrica

**Chip de ROM**
Esta EPROM de 8 K contiene el sistema operativo Wafadrive (WOS)

**Conector en paralelo**
Este conector marginal permite añadirle al sistema otras interfaces compatibles con el Spectrum





**Unidades de "wafer"**
El Wafadrive posee dos unidades de cinta, ambas autocontenidas. Rotronics ofrece para instalar en ellas cartuchos de 16, 64 y 128 K

**Motores**
Cada unidad posee su propio motor eléctrico para hacer girar la cinta en el interior de los cartuchos

**Puerta RS232**
Esta puerta en serie permite conectar el Spectrum a un equipo estándar de comunicaciones

**Interface Centronics**
El Wafadrive está equipado con una puerta Centronics que permite la adición de una impresora, necesaria para tratamiento de textos

se lo llama. A partir de entonces, el WOS se limita a comprobar si el *wafer* insertado es el mismo, examinando el siguiente sector. De ser así, el WOS visualiza el catálogo que ya tiene retenido en RAM.

Manteniéndose fiel a la idea de implementar un sistema de almacenamiento masivo que permita utilizar el Spectrum para aplicaciones más serias, Rotronics ha incluido en el paquete el programa para tratamiento de textos Spectral Writer. Se trata de un sistema relativamente amplio que hace un uso cabal del Wafadrive. A las funciones tales como reformatear un párrafo, insertar palabras y suprimir líneas se las llama utilizando la tecla Symbol Shift del Spectrum junto con otras teclas. Otras funciones que acceden a archivos retenidos en *wafer* se pueden obtener mediante la instrucción de opciones: éstas incluyen guardar (SAVE) y cargar (LOAD) archivos de texto desde cassette o desde el Wafadrive. Spectral Writer es un procesador de textos atractivo, si bien no permite que uno establezca la longitud de la línea en pantalla. Quizá sea lamentable que aún utilizando un Spectrum+, la calidad del teclado desluzca en cierto modo la eficacia del Spectral Writer.

Por supuesto, el factor de mayor peso que determina el éxito o el fracaso de todo medio de almacenamiento para ordenador es la disposición de las firmas de software para apoyarlo. Por el momento éste parece ser un grave inconveniente para el éxito del Wafadrive, puesto que ninguna de las principales firmas de software está produciendo sus programas en cartuchos Wafadrive (éste es un problema que también ha afectado a la propia Sinclair). No obstante, no todo está perdido para el usuario de un Wafadrive. Hay al menos una empresa que en la actualidad está produciendo un programa que permite que uno vuelque software comercial en cartuchos Wafadrive. Esto significa que los usuarios se verán obligados a adquirir tanto la cassette comercial como un *wafer* para efectuar en él la transferencia, pero podría ser un bajo precio pagar para obtener unos tiempos de acceso considerablemente mejores.

Otra pequeña dificultad del Wafadrive son los conectores marginales de la parte posterior de la máquina. Debido a que éstos no son estándares, los usuarios habrán de convertir las propias interfaces, o bien esforzarse por encontrar periféricos con conexiones apropiadas. Pero aunque el Wafadrive posee sus inconvenientes, es una máquina muy bien hecha y, por cierto, representa una alternativa viable a la Interface 1 y microdrive que suministra Sinclair Research.

## WAFADRIVE ROTRONICS

**DIMENSIONES**
230×110×80 mm

**INTERFACES**
Puerta en serie RS232, interfaces en paralelo Centronics, conector marginal del Spectrum

**FORMATO**
*Wafers* flexibles de bucle continuo

**CAPACIDAD**
*Wafers* de 16, 64 y 128 K

**VELOCIDAD**
Velocidad de transferencia: 16 Kbaudios; tiempo máximo de acceso: 6,5 seg (16 K), 45 seg (128 K)




**Cartucho Wafadrive**
Desde el punto de vista de las proporciones, el *wafer* se asemeja a una cassette convencional, pero la cinta que hay en su interior está empalmada formando un bucle continuo. Esto significa que no se ha de rebobinar la cinta para acceder a los datos que ya han pasado por el cabezal de lectura/escritura

Chris Stevens

# En alta mar

## Ahora que ya hemos levado anclas, debemos llevar un recuento semanal de la tripulación y el barco

Durante el viaje al Nuevo Mundo, la velocidad del barco se ve afectada por el número y fortaleza de los tripulantes. Una tripulación numerosa y saludable no contribuye a aumentar la velocidad del barco, pero cuando la tasa de fortaleza combinada disminuye por debajo de cierto punto, la ineficacia incrementa la duración del viaje. Si toda la tripulación muere, cómo es lógico, el juego se termina.

Una de las funciones del Diario de Navegación del Capitán es informar al jugador sobre la muerte de cualquier miembro de la tripulación, si se ha agotado alguna provisión, y visualizar un cálculo actualizado de la duración de lo que aún resta del viaje. En la línea 41 se crea una nueva variable, EW. Ésta representa el tiempo extra del viaje expresado en semanas (o fracciones de semana), que se sumará a las ocho semanas originales si la fortaleza acumulada de la tripulación cae por debajo de un cierto nivel. Para que la duración del viaje se pueda visualizar e incrementar en enteros sin perder la parte fraccional de EW, el tiempo extra para el viaje se calcula y se almacena en EW como un número real, pero se le suma a JL como un entero, utilizando INT.

La programación para el informe de final de semana se halla en la subrutina de la línea 5300. Habrá una gran sección del programa para tratar con las contingencias que se produzcan durante el viaje, que se irán añadiendo en futuros módulos y que comenzaremos a programar en el próximo capítulo. Estos acontecimientos incidirán también en la cantidad de provisiones, la fortaleza y el número de tripulantes y la duración del viaje y, por consiguiente, influirán en los valores impresos en la sección de este capítulo.

La primera tarea de la subrutina del informe de final de semana consiste en establecer un bucle en la línea 5325 para comprobar si alguno de los tripulantes ha fallecido. Examina el segundo elemento de la matriz de tripulación, TS(,), para cada uno de los 16 posibles tripulantes, y determina si la tasa de fortaleza está o no establecida en −999, lo que determina la muerte.

Los fallecidos se le suman a X, que es el contador semanal de las víctimas. Si el valor de la matriz no es igual a −999, se implementa el siguiente contador del bucle. Tras registrar una pérdida, la línea 5340 restablece a cero los elementos de fortaleza y categoría, para evitar que la muerte se vuelva a registrar en el siguiente informe de final de semana. Se resta entonces el marinero del total de la tripulación CN. Si no ha habido fallecimientos, el programa va a la línea 5400, donde se comprueban las provisiones. Si toda la tripulación ha muerto, el programa imprime "Toda la tripulación que quedaba ha fallecido. Su barco navega a la deriva a través del océano desierto. El juego ha terminado." Si viven todavía algunos tripulantes entonces en la línea 5390 se imprime el número de fallecidos.

Las provisiones consumidas se registran de la misma forma que los tripulantes que han perdido la vida. El programa comprueba si se ha agotado alguna provisión y prepara un bucle de 1 a 4 para representar los 4 tipos de provisiones. Si se ha agotado alguna, su valor en la matriz PA() será −999. Preparada durante la rutina "reparto de provisiones", la línea 5415 comprueba PA() e informa al jugador que "Te has quedado sin..." ese elemento en particular si en realidad el valor es de −999.

El elemento de PA() se restablece luego a 0, para impedir que al finalizar la siguiente semana se vuelva a imprimir el mensaje. Si se ha agotado más de una clase de provisiones, antes de imprimir su nombre se imprime "Y". Es posible que alguna de las provisiones agotadas se pueda conseguir en la sección de contingencias del juego. De ser así, se restablecerá el elemento de PA() para esa provisión, y figurará en todos los registros como normal.

El tiempo adicional para el viaje se calcula revisando la matriz de fortaleza de la tripulación, acumulando la fortaleza total y comparándola con la fortaleza óptima mínima. Para continuar el viaje con una eficiencia máxima, la fortaleza total de la tripulación ha de ser de al menos 800. Ello se puede conseguir con ocho tripulantes con todas sus fuerzas, o con 16 con una fortaleza media, y así sucesivamente. La línea 5465 prepara un bucle para cada elemento de la matriz de la tripulación y la 5470 acumula las fortalezas en X.

La línea 5480 comprueba si la fortaleza total es mayor que 799. De ser así, no aumentará el tiempo de duración del viaje; el programa pasa entonces a la línea 5494, que solicita que se pulse una tecla para dar comienzo a la siguiente semana.

Si la fortaleza es inferior a 800, la fórmula de la línea 5489 calcula el tiempo extra para el viaje. Le resta a 800 la fortaleza acumulada de la tripulación, divide por 800 y le suma el resultado al tiempo extra del viaje. El incremento se almacena en EW como un número real. La línea 5490 le suma la parte entera de EW a la duración del viaje, para que el tiempo de viaje restante no se visualice con fracciones de semana. El valor acumulado de EW se suma cada semana a la duración del viaje. Para que la tasa de fortaleza de un tripulante fallecido no disminuya en función de WF en la subrutina de la línea 9300 que ya hemos ofrecido, inserte esta línea:

```
9315 IF TS(S1,2)=0 THEN 9340
```

Mientras el barco vaya navegando hacia el Nuevo Mundo, varios acontecimientos influirán en la fortaleza de la tripulación. Hasta ahora, el único factor del programa que afectaba a la fortaleza era la

# Mód. 5: Diario de navegación

## Informe de final de semana

```
5300 REM INFORME DE FINAL DE SEMANA
5305 PRINTCHR$(147)
5310 S$="    DIARIO DE NAVEGACION*":GOSUB 9100
5312 S$="    ------ -- -----------*":GOSUB 9100
5314 GOSUB 9200
5316 PRINT:PRINT"            FIN DE LA SEMANA";WK
5318 GOSUB 9200
5320 X=0
5325 FOR T=1 TO 16
5330 IF TS(T,2)<>-999 THEN 5350
5335 X=X+1
5340 TS(T,1)=0:TS(T,2)=0
5345 CN=CN-1
5350 NEXT
5355 IF X=0 THEN 5400
5358 PRINT:PRINT
5360 S$="DURANTE LA SEMANA PASADA*":GOSUB 9100
5365 IF CN>0 THEN 5390
5367 S$="MURIO TODA LA TRIPULACION QUE
     QUEDABA*":GOSUB 9100
5369 GOSUB 9200:PRINT:GOSUB 9200
5375 S$="EL BARCO VA IRREMISIBLEMENTE A LA
     DERIVA*":GOSUB 9100
5374 GOSUB 9200
5376 S$="POR EL OCEANO DESIERTO...*":GOSUB 9100
5378 GOSUB 9200
5380 PRINT:PRINT
5382 S$="        EL JUEGO HA TERMINADO*":GOSUB 9100
5384 PRINT:PRINT
5386 GOSUB 9200:END
5388 GOTO 5386
5390 PRINT X;
5392 IF X=1 THEN S$="TRIPULANTE HA FALLECIDO*"
5394 IF X<>1 THEN S$="TRIPULANTES HAN FALLECIDO*"
5396 GOSUB 9100
5398 GOSUB 9200
5400 PRINT:PRINT
5405 S$="AHORA TE HAS QUEDADO SIN  "
5410 FOR T=1 TO 4
5415 IF PA(T)<>-999 THEN 5440
5420 PRINT S$;P$(T)
5425 PA(T)=0
5428 S$="                  Y"
5440 NEXT
5450 GOSUB 9200
5455 REM CALCULAR NUEVA JL
5460 X=0
5465 FOR T=1 TO 16
5470 X=X+TS(T,2)
5475 NEXT
5480 IF X>799 THEN 5494
5481 PRINT:PRINT
5482 S$="LA TRIPULACION SE HALLA POR DEBAJO DE LA
     FORTALEZA TOTAL*":GOSUB 9100
5483 S$="DE MANERA QUE EL VIAJE PODRIA DURAR
     MAS*":GOSUB 9100
5489 EW=EW+((800-X)/800)
5490 JL=JL+INT(EW)
5492 PRINT
5494 S$=K$:GOSUB 9100
5496 GET I$:IF I$="" THEN 5496
5499 RETURN
```

### Variable de las semanas extras

```
41 EW=0:REM SEMANAS EXTRAS
```

### Adición al bucle principal del viaje

```
880 GOSUB 5300:REM INFORME DE FINAL DE SEMANA
```

# Complementos al BASIC

**Spectrum:**

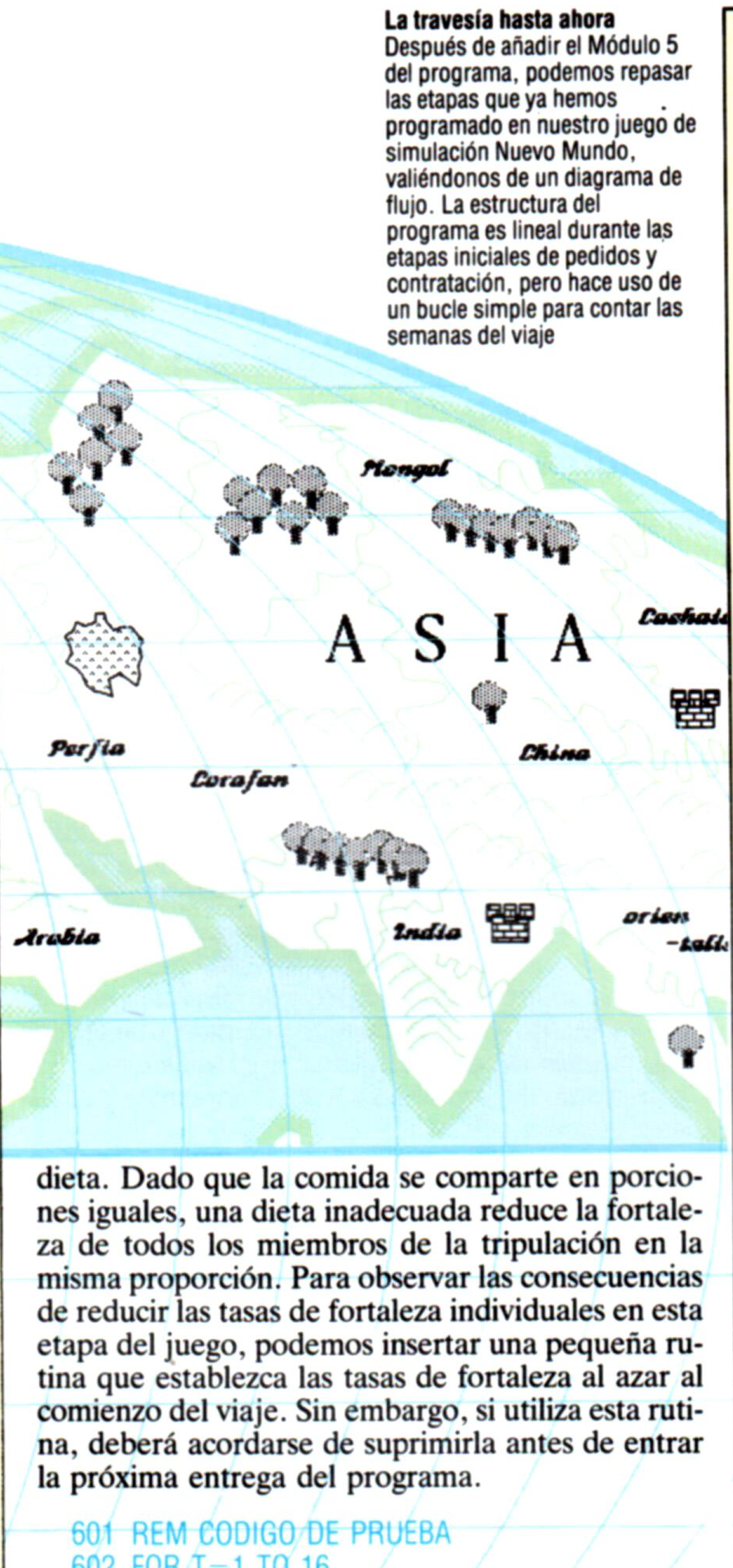
Introduzca las siguientes modificaciones:

```
5305 CLS
5496 LET I$=INKEY$:IF I$="" THEN GOTO
     5496
```

**BBC Micro:**

Introduzca estos cambios:

```
5305 CLS
5496 I$=GET$
```




dieta. Dado que la comida se comparte en porciones iguales, una dieta inadecuada reduce la fortaleza de todos los miembros de la tripulación en la misma proporción. Para observar las consecuencias de reducir las tasas de fortaleza individuales en esta etapa del juego, podemos insertar una pequeña rutina que establezca las tasas de fortaleza al azar al comienzo del viaje. Sin embargo, si utiliza esta rutina, deberá acordarse de suprimirla antes de entrar la próxima entrega del programa.

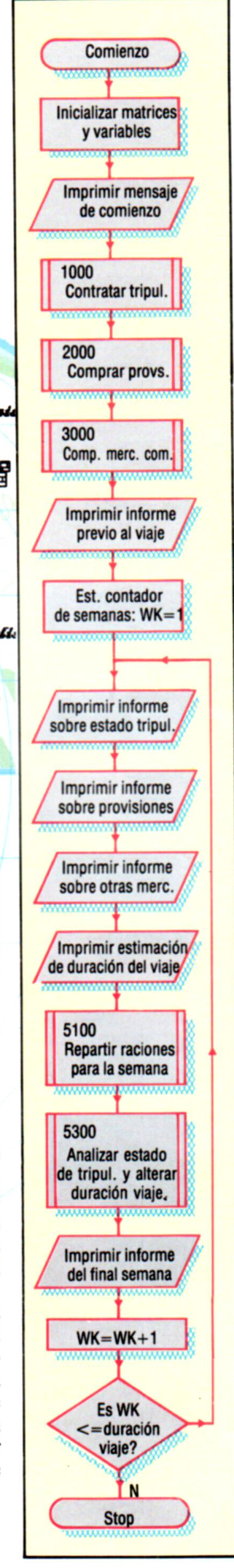
```
601 REM CODIGO DE PRUEBA
602 FOR T=1 TO 16
603 IF TS(T,2)=100 THEN TS(T,2)=
    INT(RND(1)*100)+1
604 NEXT
```

La rutina de la línea 603 revisa el segundo elemento de la matriz de la tripulación, en 16 iteraciones. Cambia toda tasa de fortaleza de 100 por un número elegido al azar entre 0 y 99, utilizando la fórmula RND dada (a cada número al azar se le suma 1 para evitar que se establezca una tasa de fortaleza en cero, lo que determinaría la muerte del tripulante).

La rutina del informe de final de semana se llama mediante una llamada GOSUB desde dentro del bucle principal del viaje, en las líneas 820-889. Originalmente, este bucle estaba controlado por una construcción FOR...NEXT, estableciendo el límite máximo la variable de la duración del viaje, JL. Sin embargo, dado que el nuevo módulo puede alterar el valor de JL, necesitamos reconstruir el bucle reemplazando las sentencias FOR y NEXT por:

```
820 WK=1:REM BUCLE PRINCIPAL DEL VIAJE
889 WK=WK+1:IF WK<=JL THEN 825
```

**La travesía hasta ahora**
Después de añadir el Módulo 5 del programa, podemos repasar las etapas que ya hemos programado en nuestro juego de simulación Nuevo Mundo, valiéndonos de un diagrama de flujo. La estructura del programa es lineal durante las etapas iniciales de pedidos y contratación, pero hace uso de un bucle simple para contar las semanas del viaje




Ian McKinnell

# Registros

## En PASCAL, los registros representan la forma más útil de empaquetar juntos datos de distinto tipo

En el capítulo anterior, le mostramos cómo los conjuntos pueden ser sumamente útiles para el programador de PASCAL, y que se pueden implementar con eficacia a nivel máquina porque las operaciones definidas para ellos poseen equivalentes directos en los juegos de instrucciones nativos de la mayoría de los procesadores. Siempre se construyen reuniendo un grupo de elementos que pueden ser de cualquier tipo escalar (no real). Sin embargo, aparte de poderse comprobar la pertenencia mediante el operador IN, todas las operaciones de conjuntos están definidas sobre la totalidad de la estructura. Si bien no hay ningún mecanismo de selección para extraer un elemento en particular, podemos utilizar los operadores de inclusión de conjunto (<=>= ) y comprobar la equivalencia de conjuntos ("igualdad") con = y <>.

Cuando los conjuntos los manipulan matemáticos, los mismos no sólo pueden contener objetos ilimitados, sino que también se los puede considerar teóricamente infinitos (tales son las ventajas de resolver problemas en un trozo de papel, respecto a un hardware físicamente limitado). La implementación que usted posee pondrá algún límite tanto al tamaño como a la gama de los conjuntos, de modo que no intente definir tipos ambiciosos como:

```
GranConj=SET OF enteros;
```

El límite inferior de un elemento perteneciente a un conjunto debe tener un valor ordinal de cero o mayor, y el límite superior estará restringido a algún valor absoluto: entre 255 y 4 095 en implementaciones de microordenador. Observe que el conjunto vacío (representado literalmente mediante []) es miembro de todos los conjuntos posibles, sea cual sea su tipo. Esto podría parecer un agujero en la sólida armadura tipológica del PASCAL, pero, en la práctica, el tipo de conjunto vacío siempre se puede deducir a partir del tipo de otros conjuntos de cualquier expresión.

La inclusión estricta de subconjuntos verdaderos no se puede obtener directamente, y los operadores < y > no están definidos para estructuras de conjunto. Ello se debe a razones de implementación: la mayoría de las restricciones del PASCAL tienen sentido desde el punto de vista de la eficacia o la lógica. Si usted deseara comprobar una inclusión estricta, sería necesaria una comparación doble, por ejemplo:

```
(A>=B) AND (A<>B)
```

La prueba para House en nuestro programa *Bingo* no exigía esta comparación especial, puesto que un conjunto de números cantados puede que sea equivalente ("igual") o bien, lo que es más probable, un subconjunto de los números del cartón. De modo que, por ejemplo, esta expresión booleana se podría haber expresado como:

```
House :=Cantando>=Carton
```

que resulta verdadera cuando todos los miembros de Carton están incluidos en el conjunto de los números Cantado. Ésta es exactamente la clase de propiedad fundamental de los conjuntos que los convierte en valiosísimas estructuras de datos para la resolución de problemas. Por ahora, probablemente la aplicación más útil de los conjuntos será para la comprobación de subconjuntos del tipo char. El siguiente esquema (en PASCAL/castellano) sería de utilidad para programar cualquier aplicación interactiva, como por ejemplo un juego.

```
Negativo :=['N','n'];
Afirmativo :=['S','s'];

REPEAT
  {Desarrollar el juego}
  {visualizar el(los)marcador(es)}
  write ('Otra vez?');
  ReadLn (respuesta);

  WHILE NOT (respuesta IN Afirmativo+
  Negativo)DO
    BEGIN
      WriteLn('S(i) o N(o)':Columnas);
      write ('Otra partida?');
      ReadLn (respuesta)
    END

UNTIL respuesta IN Negativo
```

Cuando deseamos acceder a elementos individuales de una estructura, las alternativas de que disponemos son matrices (suficientemente familiares), archivos y registros. Estos últimos poseen la eficaz habilidad de reproducir registros de datos de la vida real, con "campos" mezclados de cualquier tipo de datos: simples o estructurados.

La forma de registro de datos más común que se utiliza en el campo de gestión contiene campos para nombres, direcciones, números de teléfono, códigos de cuentas, etc. Lo importante es poder manipular estos datos como un bloque de información y *también* poder acceder a cualquier campo individual y procesar adecuadamente los datos. El PASCAL permite asignar tales objetos (y manipularlos de otras maneras) como un todo, permitiendo, al mismo tiempo, acceder a cualquiera de los campos componentes con fines de comparación o proceso, en función de su tipo de datos individual. La definición de un registro mixto es bastante directa:

```
TYPE
  habitacion=RECORD
        numero  : 1..999;
        ala     : (Norte,Este,Sur,Oeste);
        ocupada : boolean
    END;{habitacion}
VAR
  servicio:habitacion;
```

Así como la sentencia CASE era excepcional en el uso de la palabra reservada END como delimitador,



la definición de un registro constituye la única excepción, en la parte de declaración de un bloque, a la regla de que los BEGIN y END se utilizan de a pares. Por este motivo resulta útil complementar el END de un registro con su identificador, como en nuestro ejemplo. Cualquier variable de tipo habitacion contendrá tres campos componentes. En este caso, cada campo es de un tipo escalar distinto, pero podrían igualmente ser del mismo tipo, simple o estructurado. No existe restricción en cuanto a los tipos permitidos dentro de los campos de un registro, ¡de modo que podemos tener un campo que sea una matriz de archivos de registros que contengan conjuntos!

Dentro de las palabras delimitadoras RECORD y END, la sintaxis de definición es exactamente igual que en una declaración VAR. Aquí, sin embargo, estamos declarando identificadores de campo que son una parte integral de la estructura del registro. Por ello los nombres numero, ala y ocupada no existen fuera del "ámbito" del identificador del registro. Estos nombres de campo son identificadores locales y podrían duplicar los nombres de variables del programa. Sólo se puede acceder a ellos a través de dos mecanismos que posee el PASCAL para seleccionar registros: la notación de "punto" y la sentencia WITH.

Para seleccionar un campo determinado utilizando la notación de punto, se separa el identificador de todo el registro del identificador de campo siguiente mediante un punto ("."). Por ejemplo:

```
servicio.numero
```

aludiría solamente al campo del subrango de enteros. Podríamos inicializar el contenido del registro con sentencias tales como:

```
read (servicio.numero);
'servicio.ala:=Este;'
'servicio.ocupada:=persona<>[]'
```

etcétera. Observe que nos hemos introducido en un conjunto vacío con un tipo que depende del tipo, cualquiera que sea, de persona, ¡y ni siquiera lo hemos verdaderamente declarado!

## La sentencia WITH

La notación de punto se puede volver algo engorrosa en los casos en que deseemos acceder a la mayoría o la totalidad de los campos de un registro. Existe una sentencia estructurada alternativa que "descubre" los identificadores de campo. La semántica de la sentencia WITH significa, en líneas generales: "Quiero hacer algo con este registro, de modo que deseo especificar sólo los nombres de los campos." La sintaxis de la sentencia WITH tiene exactamente la misma forma que el bucle WHILE, y la secuencia de asignaciones de inicialización que hemos dado se podría expresar mejor de este modo:

```
WITH servicio DO
  BEGIN
    numero:=123;
    ala:=Este;
    ocupada:=true
  END
```

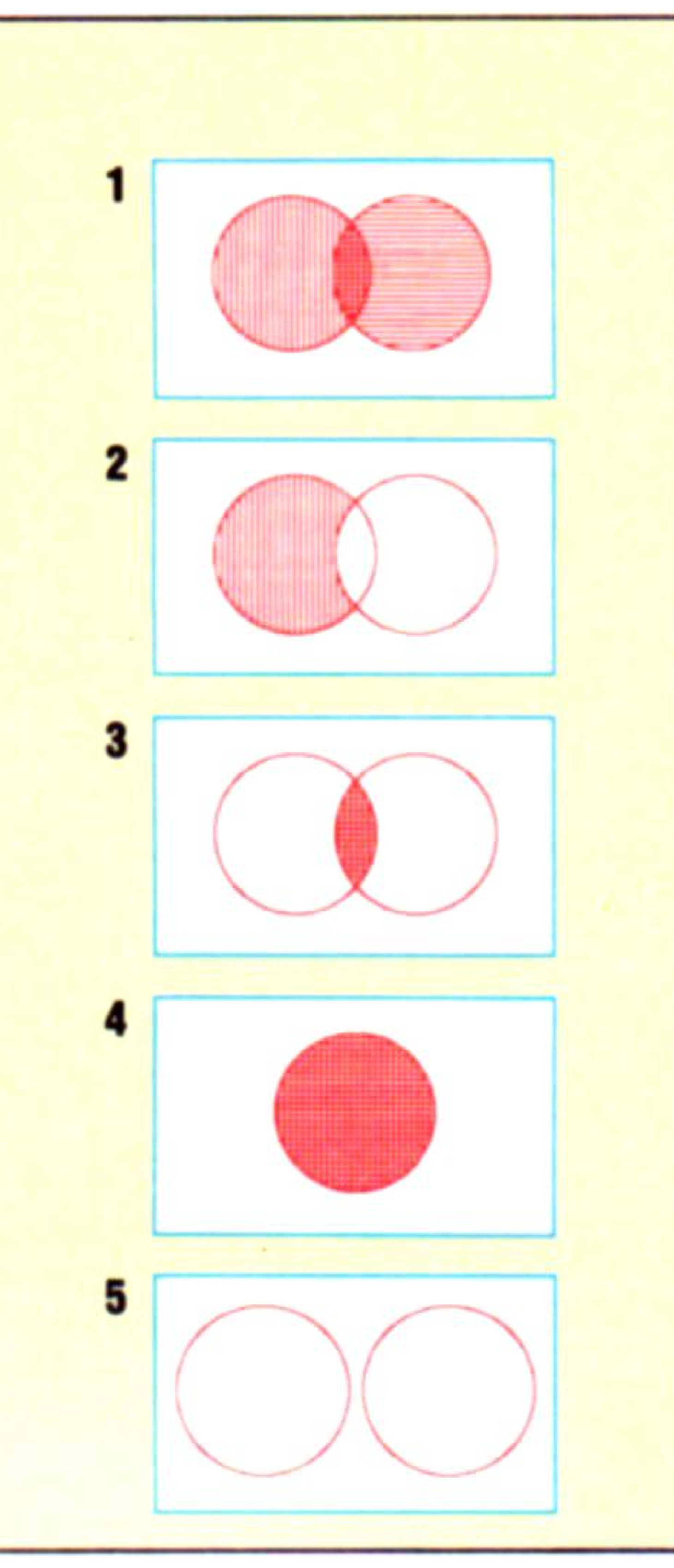
A lo largo de la sentencia WITH (desde el BEGIN al END, en este caso), no se han de cualificar los identificadores de campo con el identificador del registro y el punto, y se puede aludir a ellos directamente, como podemos ver. Aquí no habría ninguna confusión si hubiera otra variable del programa denominada Numero, por ejemplo. El ámbito local siempre tiene precedencia. Sin embargo, utilizando la notación de punto podemos comunicar valores a través de estos límites del ámbito. A modo de ejemplo, tomemos:

```
servicio.numero:=Numero
```

## Operaciones de conjuntos

Ofrecemos una lista de todas las operaciones sobre conjuntos disponibles en PASCAL (con diagramas de Venn en los casos apropiados)

**1.** *Unión* (C1+C2): Es el superconjunto que comprende a todos los objetos pertenecientes tanto al conj. C1 como al conj. C2 combinados (diag. 1)
**2.** *Diferencia* (C1−C2): El subconjunto de los miembros de C1 que no pertenecen a C2 (diag. 2)
**3.** *Intersección* (C1*C2): El subconjunto de los objetos comunes a ambos conjuntos (diag. 3)
**4.** *Equivalencia* (C1=C2): Donde los miembros pertenecientes a C1 y C2 son idénticos (diag. 4)
**5.** *No equivalencia* (C1<>C2): Es verdadera cuando ninguno de los miembros de C1 es miembro de C2, y viceversa (diagrama 5)
**6.** *Inclusión* (C1<=C2): Es verdadera si todos los miembros de C1 son también miembros de C2
**7.** *Inclusión* (C1>=C2): Es verdadera si todos los miembros de C2 son también miembros de C1
**8.** *Pertenencia* (m IN C1): Es verdadera si el miembro de un conjunto de un solo elemento ([m]) es un subconjunto de C1




Liz Dixon

## El máximo "break"

Ésta es nuestra solución al problema que planteamos en el capítulo anterior, en el que le pedíamos que creara un programa que calculara el máximo *break* posible en una partida de snooker

```
PROGRAM     MaxBreak     (output);

CONST
      BolasRojas    = 15;

TYPE
      snooker       = (blanca,roja,amarilla,verde,
                        marron,azul,rosa,negra);
VAR
      bola          :snooker;
      marcador      :BolasRojas..MaxInt;
BEGIN
  marcador :=BolasRojas*(ord (roja)+ord(negra));

  FOR bola := amarilla TO negra DO
    marcador := marcador+ord (bola);

  WriteLn ('El break maximo es :', marcador :3)
END.
```

Ian McKinnell

que le asignaría el valor de una variable externa (Numero) al campo del registro (numero). Recuerde que el PASCAL no hace distinciones entre tipos de letras (es decir, no hace ninguna diferencia entre letras en mayúscula y letras en minúscula), y el identificador no cualificado numero aludiría a una variable externa, a menos que se hallara dentro de una sentencia WITH. En todo caso, sería imprescindible utilizar la notación de punto si tuviéramos una asignación entre dos variables del mismo tipo de registro, supongamos:

```
grupos.ala:=recepcion.ala
```

Aquí la sentencia WITH sólo se podría emplear para descubrir uno de los campos de la variable, porque de lo contrario se produciría una confusa ambigüedad que no sería aceptada por el compilador.

La decisión más acertada que podríamos adoptar sería la siguiente:

```
WITH grupos DO
  ala:=recepcion.ala
```

En consecuencia, las dos formas de notación tienen su uso propio. La forma que se ha de utilizar resulta obvia a partir de la aplicación.

## Un largo recorrido

El programa *Distancias* lee desde el teclado dos distancias expresadas en yardas, pies y pulgadas. Se utiliza una definición de tipo de registro para asignar un campo separado para cada valor de las unidades de medida diferentes. El programa suma luego las longitudes entre sí, pasando las pulgadas al campo "pies" y los pies de más al campo "yardas", mediante el empleo de los operadores de enteros DIV y MOD para dar la suma y el resto, respectivamente. Los resultados se asignan a los campos de "Total" y después se imprimen. La asignación de registros enteros sería directa (p. ej.: LongA:=LongB), pero todo proceso o manipulación se debe aplicar a cada uno de los campos componentes, de modo que Total:=LongA+LongB es ilegal. Observe el empleo de la notación de punto y la sentencia WITH para leer los valores de LongA y LongB.

```
PROGRAM        Distancias        (input, output);
CONST
          MaxByte       = 255;
TYPE
          byte          = 0..Maxbyte;

          distancia     = RECORD
                             rpulgadas    : 0..11;
                             rpies        : 0..2;
                             ryardas      : byte
                          END;{distancia}
VAR
            pulgadas,
            pies        : byte;
            LongA,
            LongB,
            Total       : distancia;
BEGIN
   WriteLn ('Entre las longitudes en pulgadas pies y yardas,');
   WriteLn ('separadas por SPACE o RETURN.');
   WriteLn;
   write ('Primera:':15);
   read (LongA.ryardas,LongA.rpies, LongA.rpulgadas);
   write ('Segunda:':15);

   WITH LongB DO
      read (ryardas,rpies,rpulgadas);

   pulgadas:=LongA.rpulgadas+LongB.rpulgadas;
   pies:=LongA.rpies+LongB.rpies+pulgadas DIV 12;
   WriteLn;

   WITH Total DO
     BEGIN
       rpulgadas:=pulgadas MOD 12;
       rpies:=pies MOD 3;
       ryardas:=LongA.ryardas+LongB.
                  ryardas+pies DIV 3;
       WriteLn ('La longitud total es:');
       WriteLn (ryardas:25,'yardas',rpies:
                1,'piesy',rpulgadas:1,'pulgadas')
     END
END
```

## Protección de datos

A partir de nuestro programa *Bingo* le habíamos propuesto a modo de ejercicio que le añadiera al programa construcciones de bucle para impedir la repetición de un número cantado y detectar cualquier número ilegal fuera de la escala entre 1 y 90. La mejor forma de proteger al programa de la entrada de datos equivocados consiste en añadir una construcción WHILE tras las sentencias ReadLn. La estructura será la misma en cada caso:

```
WHILE el número sea inaceptable DO
   dar un mensaje de error apropiado
   escribir otra petición
   volver a leer los datos
```

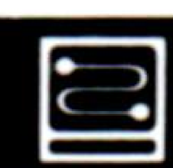
En el caso de las entradas iniciales para el cartón, debemos asegurarnos de que el número entrado no esté fuera de la escala permitida (de 1 a 90), y también evitar la repetición de una entrada previa. Por consiguiente:

```
WHILE NOT (numero IN Permitido)OR(numero
IN Carton)DO
   BEGIN
      WriteLn (numero:20,'no es valido');
      write ('Vuelva a entrar:':14);
      ReadLn (numero)
   END;
      {etc.}
```

Para cantar los números se aplica la misma estrategia, pero con la condición para entrar en el bucle de comprobación alterada:

```
WHILE NOT (numero IN Permitido)OR
(numero IN Cantado)DO
   {etc.}
```

Ian McKinnell



# La vuelta al bloque del Commodore 64

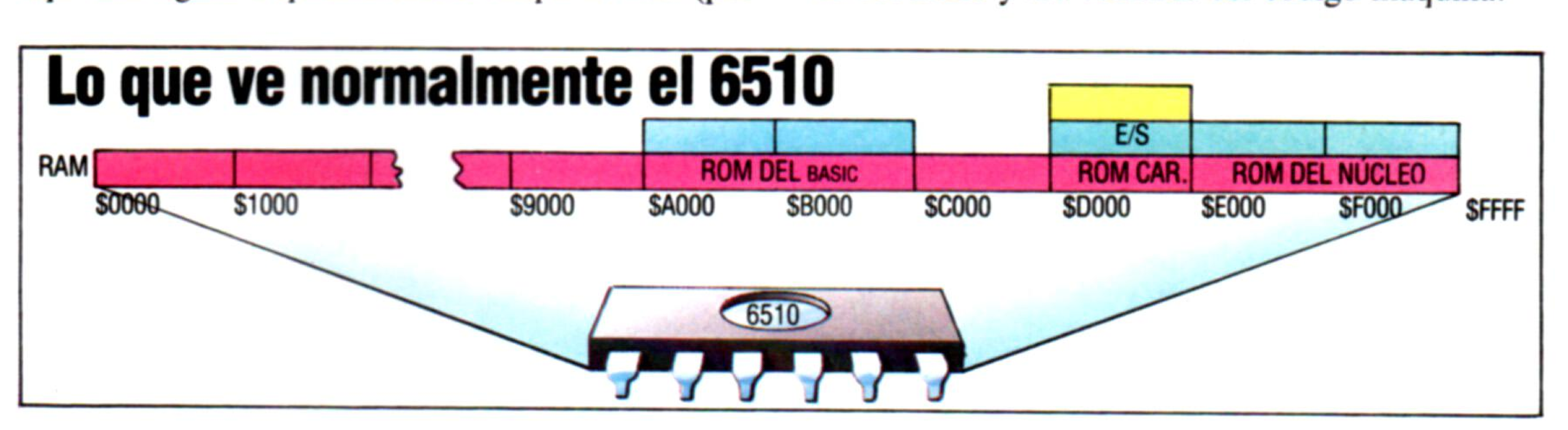
**Iniciamos una serie de capítulos dedicados al estudio de las aplicaciones prácticas ofrecidas por el OS del Commodore 64**

La primera cosa que necesitamos saber sobre el ordenador desde un punto de vista de programación de sistema es la estructuración general del mapa de la memoria. En el caso del Commodore 64 hay que decir que el microprocesador 6510 puede "ver" uno entre "varios" mapas, según cómo el programador configure la máquina. Esto se debe a que el 6510 es un microprocesador muy versátil con posibilidad de conmutar la conexión con diferentes bloques de memoria. Esto lo consigue el hardware poniendo las patillas 8 y 9 de la puerta de ampliación arriba (+5 V) o abajo (0 V), o bien, el software, alterando el contenido de la dirección 1. Cualquiera de estos dos procedimientos alterará radicalmente la manera en que el 6510 ve la memoria. Ofrecemos aquí una figura esquemática del mapa normal (por defecto), mostrando los 64 K de memoria que el 6510 ve normalmente. La tabla adjunta hace un detallado reconocimiento del mapa de la memoria.

De momento, supondremos que el Commodore 64 está en el modo por defecto. Lo primero que hay que remarcar es que el ordenador tiene 64 K de RAM y 20 K de ROM. De este total de 84 K, el 6510 sólo puede ver cada vez un máximo de 64 K, de aquí la necesidad de emplear la facilidad conmutación de la conexión. Por el diagrama del mapa de la memoria se puede observar que la ROM del intérprete del BASIC (el programa en código máquina que ejecuta los programas en BASIC) se encuentra entre $A000 y $CFFF, y la ROM núcleo (el código que maneja todas las funciones de entrada/salida) está entre $E000 y $FFFF.

Más abajo de estos dos bloques de ROM se hallan dos bloques de RAM de ocho K, que el 6510 no ve en modo normal, aunque una POKE (o STA) referida a estas direcciones "las atravesará" hasta llegar a la RAM subyacente. Una PEEK nos dará el contenido de la ROM superpuesta, pero la RAM puede ser leída si "desconectamos" la ROM correspondiente. Hay que insistir en que si usted desea emplear las instrucciones PEEK o POKE del BASIC, será mejor que no "desconecte" la ROM del BASIC.

Lo mismo que todo programa en BASIC necesita una RAM donde almacenar sus variables, así las rutinas del intérprete y del núcleo necesitan una RAM del OS. La RAM del OS se encuentra en las posiciones entre $0002 y $03FF, y uno de sus componentes importantes, la pila, ocupa las posiciones entre $0100 y $01FF. La RAM del OS se coloca en la página cero porque forma parte del mapa de la memoria que permite el acceso más rápido a la RAM, y la velocidad de la operación es un factor muy importante en el OS. Si intenta usted interrelacionar un programa en BASIC con algún tipo de código máquina, es vital que posea un conocimiento práctico de esta área de la RAM. Veamos dos aspectos importantes de la RAM del OS: los punteros del BASIC y los vectores del código máquina.




**ROM a la vista**
Dado que el procesador 6510 emplea direcciones de 16 bits, sólo puede direccionar un máximo de 65 536 (64 K) posiciones de memoria. El Commodore 64 tiene un total de 84 K de ROM y RAM, pero resuelve este problema de su limitación "óptica" a sólo 64 K cada vez conmutando la conexión a distintas zonas de memoria.
El "bloque" frontal representa la memoria vista normalmente por el 6510. A las áreas que quedan detrás de él se puede acceder estableciendo los registros especiales de conexión. Por ejemplo, si se escribe un programa sólo en lenguaje máquina, la ROM del BASIC probablemente no es necesaria y se puede "cegar" para que pueda ser usada la RAM de la parte trasera

## Análisis del mapa de memoria del C 64

| Posiciones | Empleo |
|---|---|
| $0000-$0001 | Registros de control del mapa de mem. del 6510 |
| $0002-$03FF | RAM del sistema operativo |
| $0400-$07F7 | RAM de pantalla |
| $07F8-$07FF | Punteros de sprites |
| $0800-$9FFF | Área de progs. en BASIC usuario (incl. variables) |
| $A000-$BFFF | ROM del intérprete de BASIC |
| $C000-$CFFF | 4 K de RAM libre |
| $D000-$D02E | Registros de control del chip VIC II (6566/9) |
| $D02F-$D3FF | Repetición de imágenes del VIC II |
| $D400-$D41C | Registros de control del chip SID (6581) |
| $D41D-$D7FF | Repetición de imágenes del SID |
| $D800-$DBE7 | Cuartetos de color (i.e., cuatro bits) |
| $DBE8-$DBFF | Cuartetos sin usar |
| $DC00-$DC0F | Registros de control del chip CIA #1 E/S (6526) |
| $DC10-$DCFF | Repetición de imágenes de CIA #1 |
| $DD00-$DD0F | Registros de control del chip CIA #2 E/S (6526) |
| $DD10-$DDFF | Repetición de imágenes de CIA #2 |
| $E000-$FFFF | ROM del núcleo |

## Punteros del BASIC

Un cambio en el contenido de los punteros del BASIC puede resultar útil a veces. Por ejemplo, el contenido normal de $002B (43, en decimal) y $002C (44, en decimal) son, respectivamente, 1 y 8. Ésta es la dirección del inicio del BASIC retenida en forma *lo-hi*, y significa que el BASIC comienza en 8×256+1=2049, es decir, en $0801. El BASIC se inicia, efectivamente, en $0800, pero el OS exige que el primer byte sea siempre cero, de modo que el inicio propio de un programa en BASIC será en $0801. ($0801 es la dirección donde se inicia la operación de guardar un programa en BASIC cuando se emplea un monitor Commodore de cód. máq.)

Antes de cargar cualquier programa en BASIC es posible cambiar el límite inferior del BASIC manipulando estos dos punteros, llevando buen cuidado de que el BASIC se inicie en una frontera de página (o sea, hay que guardar el contenido de la posición $002B en uno de ellos) y que comience con un cero. (Obsérvese también que una "página" es un bloque de 256 bytes de memoria.) Así:

```
POKE2560,0:POKE44,10:NEW
```

ejecutado en el modo directo hará subir el límite inferior de la memoria dos páginas más arriba hasta $2560. Aquí NEW se usa para restablecer rápidamente todos los punteros (que ocupan las posiciones $002D a $0038). El elevar el límite inferior del BASIC puede servir para albergar en la memoria dos programas en BASIC simultáneamente. Basta con cargar el primer programa, elevar el límite inferior y después cargar el segundo.

Con más frecuencia, quizá, usted habrá de bajar la parte superior de la memoria para conseguir más espacio para el programa en leng. máq. Así,

```
POKE56,159:POKE51,0:POKE52,159
```

hará descender una página entera ese límite superior de la memoria para BASIC.

Una vez rescatado un bloque de RAM del dominio del BASIC podemos estar seguros de que el OS no empleará esta área para almacenar variables del BASIC, y por ello estará protegido nuestro código máquina (salvo de nuestros propios errores).

### Tabla punteros BASIC EN CBM 64

| Posiciones | Puntero |
|---|---|
| $002B-$002C | Inicio BASIC del usuario |
| $002D-$002E | Inicio de variables BASIC |
| $002F-$0030 | Inicio de vectores BASIC |
| $0031-$0032 | Fin de vectore BASIC+1 |
| $0033-$0034 | Parte inf. de las series |
| $0035-$0036 | Utilidad |
| $0037-$0038 | Fin BASIC del usuario |

## Los vectores de la RAM

El segundo bloque de RAM que reviste un interés particular para el programador es el bloque situado entre $0314 y $0333, el cual contiene los vectores de la RAM. Estos vectores son como un desvío opcional a una línea secundaria en un ferrocarril; por lo general, cuando un tren (es decir, el 6510 que ejecuta su programa, siguiendo el símil) sigue una línea pasa por el interruptor sin desviar su dirección. Pero a veces conviene desviar el tren por otra línea lateral pasando por una o dos estaciones antes de reemprender su ruta normal.

Usaremos como ejemplo el vector IRQ (solicitud de interrupción). Cada sesentavo de segundo, en un Commodore que opera normalmente, se dispara uno de los temporizadores del chip de E/S 6526 y baja la línea IRQ del 6510. Al final de su instrucción en curso, el 6510 responderá a la bajada de la línea IRQ generando una interrupción y comenzando la rutina de servicio de la IRQ, fragmento de código de mantenimiento que se inicia en $FF48. Entre otras cosas, la rutina de servicio inspecciona el teclado para detectar la pulsación de cualquier tecla. Una de las primeras cosas que hace la rutina es:

```
JMP ($0314)
```

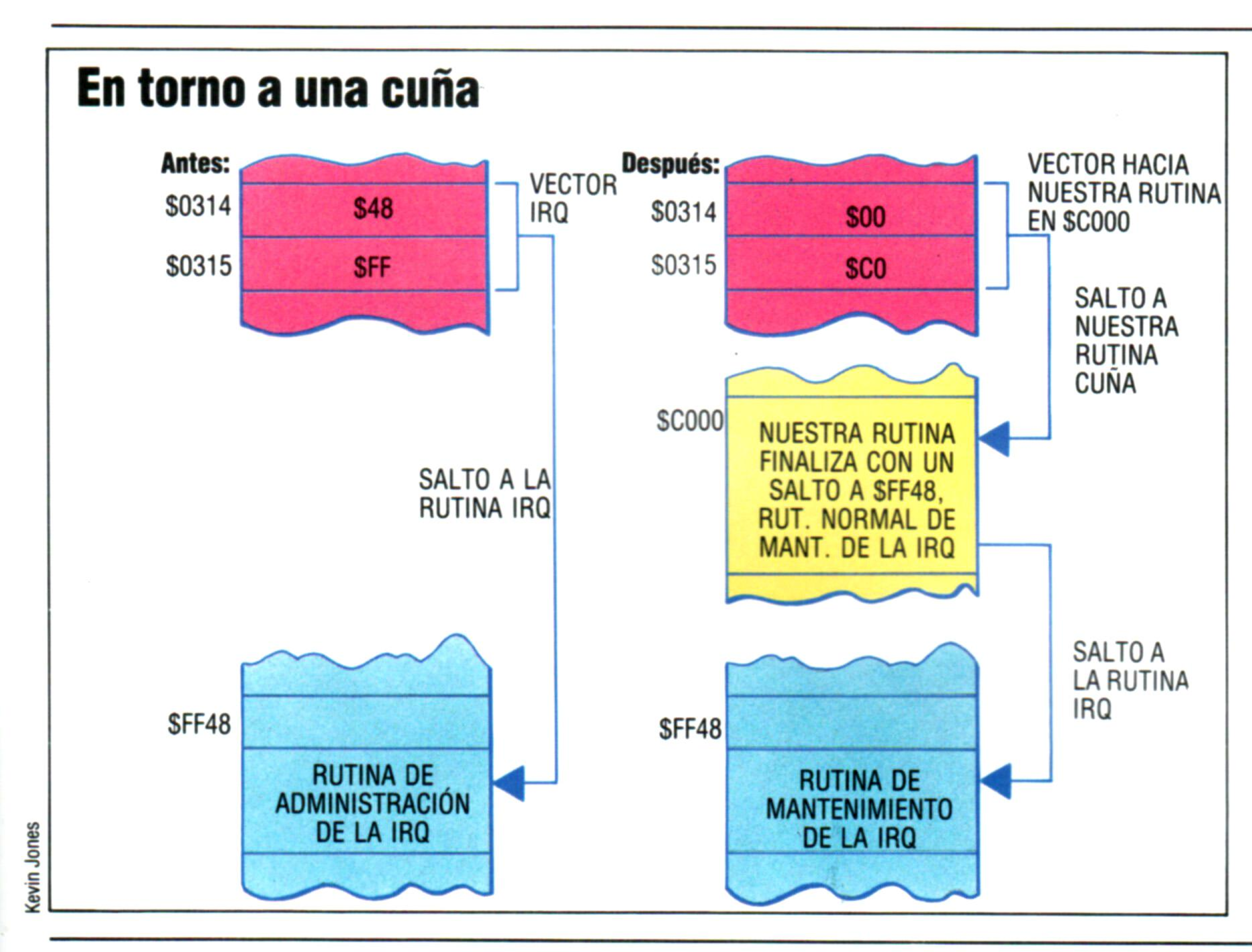
o sea, realiza un salto indirecto a la dirección contenida en las posiciones $0314 (byte *lo*) y $0315 (byte *hi*). Estos dos bytes están en la RAM, por lo que se comienza a entrever que si cambiamos su contenido podremos apuntar a nuestra propia sección de código. Una vez ejecutado nuestro código podemos hacer que el microprocesador vuelva a apuntar a la dirección que tenía anteriormente. Así, y siempre que nuestro código no sea demasiado pretencioso (lo que excluye el uso de rutinas de núcleo) podemos conseguir que el 6510 nos realice una pequeña rutina cada sesentavo de segundo. Tal fragmento de código se denomina una *cuña*.

### Tabla de vectores en el CBM 64

| Posiciones | Vector |
|---|---|
| $0314-$0315 | Interrupción IRQ |
| $0316-$0317 | Interrupción BRK |
| $0318-$0319 | Interrupción NMI |
| $031A-$031B | Rutina OPEN del núcleo |
| $031C-$031D | Rutina CLOSE del núcleo |
| $031E-$031F | Rutina CHKIN del núcleo |
| $0320-$0321 | Rutina CHKOUT del núcl. |
| $0322-$0323 | Rutina CLRCHN del núcl. |
| $0324-$0325 | Rutina CHRIN del núcleo |
| $0326-$0327 | Rutina CHROUT del núcl. |
| $0328-$0329 | Rutina STOP del núcleo |
| $032A-$032B | Rutina GETIN del núcleo |
| $032C-$032D | Rutina CLALL del núcleo |
| $032E-$032F | Definido por el usuario |
| $0330-$0331 | Rutina LOAD del núcleo |
| $0332-$0333 | Rutina SAVE del núcleo |



## En torno a una cuña




**La cuña de tiempo**
Cada sesentavo de segundo es interrumpido el procesador 6510 para realizar tareas de mantenimiento tales como inspeccionar el teclado. La dirección de inicio de la rutina de mantenimiento está contenida en un par de posiciones, la $0314 y la $0315. Si se cambia la dirección contenida en estas posiciones, es posible "incrustar" nuestro propio fragmento de código máquina, que será ejecutado en lugar de la anterior rutina. Tal fragmento suele acabar con un salto a la rutina normal de mantenimiento. De este modo podemos estar seguros de que nuestro fragmento en código máquina será ejecutado cada sesentavo de segundo, al generarse la interrupción

## El reloj Commodore

El siguiente listado en assembly y programa cargador de BASIC ilustra la teoría de cómo insertar una cuña o fragmento de código dentro de la rutina normal de tratamiento de interrupción IRQ. Esta cuña lleva "un reloj que da las horas" y lo visualiza en una esquina de la pantalla. El reloj, cada sesentavo de segundo, es modificado por la interrupción normal, de modo que la máquina sigue operando normalmente con el reloj en marcha, lo que permite entrar un programa en BASIC y ejecutarlo simultáneamente. La rutina establece ante todo el tiempo inicial que se le pasa, y después cambia el contenido de $0314 y $0315 para que apunten al código cuña, el cual se encarga de actualizar y visualizar el reloj cada vez que se genera una interrupción IRQ. El trabajo final consiste en realizar un salto (JMP) a una rutina normal de tratamiento de la IRQ, cuya dirección se guarda en las posiciones VECTOR y VECTOR+1.

Una alternativa para entrar y ensamblar este código fuente sería entrar el programa como una serie de sentencias DATA que constituyen el programa en código máquina. Se entra sencillamente el programa cargador de BASIC y se ejecuta para ver cómo aparece el reloj en el extremo superior derecho de la pantalla.

Se puede poner el reloj a la hora, minutos y segundos que se desea mediante POKE, que introduce la información en las posiciones 50129,50130 y 50131. Al final del programa cargador de BASIC el reloj se establece mediante las líneas 1480 y siguientes en la hora inicial 4:30:00 pm. Basta cambiar estos valores introducidos (POKE) en estas posiciones para obtener la hora que usted quiera. SYS 50138 hace que se ejecute el programa en código máquina y que empiece a funcionar el reloj. Se puede parar el reloj apretando las teclas RUN/STOP y RESTORE a la vez, o ejecutando una rutina especial que se incluye dentro del programa entrando SYS 50237

**Cargador de BASIC**

```
1000 REM ** CARGADOR EN BASIC DEL RELOJ **
1010 DATA173,166,2,240,10,169,128,13,14
1020 DATA221,141,14,221,48,8,169,127,45
1030 DATA14,221,141,14,221,169,127,45
1040 DATA15,221,141,15,221,173,208,195
1050 DATA41,128,141,208,195,173,209,195
1060 DATA32,28,197,13,208,195,141,11
1070 DATA221,173,210,195,32,28,197,141
1080 DATA10,221,173,211,195,32,28,197
1090 DATA141,9,221,169,0,141,8,221,120
1100 DATA173,20,3,141,214,195,173,21,3
1110 DATA141,215,195,169,76,141,20,3
1120 DATA169,196,141,21,3,88,96,120,173
1130 DATA214,195,141,20,3,173,215,195
1140 DATA141,21,3,88,96,173,216,195,201
1150 DATA6,240,3,76,8,197,169,255,141
1160 DATA216,195,173,213,195,240,243
1170 DATA173,11,221,170,41,128,208,5
1180 DATA169,1,76,111,196,169,16,141,38
1190 DATA4,173,212,195,141,38,216,163
1200 DATA13,141,39,4,173,212,195,141,39
1210 DATA216,138,41,16,32,14,197,141,28
1220 DATA4,173,212,195,141,28,216,138
1230 DATA32,22,197,141,29,4,173,212,195
1240 DATA141,29,216,169,58,141,30,4,173
1250 DATA212,195,141,30,216,173,10,221
1260 DATA170,32,14,197,141,31,4,173,212
1270 DATA195,141,31,216,138,32,22,197
1280 DATA141,32,4,173,212,195,141,32
1290 DATA216,169,47,141,33,4,173,212
1300 DATA195,141,33,216,173,9,221,170
1310 DATA32,14,197,141,34,4,173,212,195
```

```
1320 DATA141,34,216,138,32,22,197,141
1330 DATA35,4,173,212,195,141,35,216
1340 DATA169,46,141,36,4,173,212,195
1350 DATA141,36,216,173,8,221,105,48
1360 DATA141,37,4,173,212,195,141,37
1370 DATA216,238,216,195,108,214,195,74
1380 DATA74,74,74,24,105,48,96,41,15,24
1390 DATA105,48,96,160,255,56,200,233
1400 DATA10,176,251,105,10,141,217,195
1410 DATA152,10,10,10,10,13,217,195,96
1420 DATA42131:REM*CHECKSUM*
1430 CC=0
1440 FORI=50138TO50481
1450 READX:CC=CC+X:POKEI,X
1460 NEXT
1470 READX:IFCC<>XTHENPRINT"CHECKSUM ERROR"
1480 REM ** TEST CLOCK **
1490 POKE50128,128:REM AM/PM
1500 POKE50132,8:REM COLOR
1510 POKE50133,1:REM VISUALIZACION
1520 POKE50129,4:REM HORAS
1530 POKE50130,30:REM MINUTOS
1540 POKE50131,0:REM SEGUNDOS
1550 SYS50138:REM RUTINA DE LLAMADA
```

PARA DESCONECTAR LA CUÑA EMPLEAR SYS50237
EN MODO DIRECTO O DE PROGRAMA

**Listado assembly**

```
;**********************************
;*          CUÑA IRQ RELOJ        *
;*                                *
;*                                *
;* 50128=AM=0/PM=128              *
;* 50129=HORAS                    *
;* 50130=MINUTOS                  *
;* 50131=SEGUNDOS                 *
;* 50132=COLOR DEL RELOJ          *
;* 50133=VISUAL ON=1/OFF=0        *
;*                                *
;* INSERCION CUÑA SYS 50138       *
;* REMOVER CUÑA SYS 50237         *
;**********************************
;
IRQVEC=$0314        ; VECTOR RAM IRQ
CLOCK=$DD08         ; REGISTRO TOD
D2CRA=$DD0E         ; VIA#2 CRA
D2CRB=$DD0F         ; VIA#2 CRB
PALNTS=$02A6        ; FLAG PAL/NTSC
RATE=$06            ; VISUALIZA CADA 6 IRQ
DIGIT=$30           ; CODIGO PANT. DE '0'
POINT=$2E           ; CODIGO PANT. DE '.'
SLASH=$2F           ; CODIGO PANT. DE '/'
COLON=$3A           ; CODIGO PANT. DE ':'
HZ50=$80            ; MASC. DE 50 HZ PARA TODIN
HZ60=$7F            ; MASC. DE 60 HZ PARA TODIN
AY=$01              ; CODIGO PANT. DE 'A'
PEE=$10             ; CODIGO PANT. DE 'P'
EM=$0D              ; CODIGO PANT. DE 'M'
WRITE=127           ; MASC. PARA PONER RELOJ EN CRB
SCNLOC=$041C        ; DIR. RELOJ EN PANT.
COLLOC=$D81C        ; DIR. EN MATRIZ VIDEO
TRNCLO=$0F          ; MASC. PARA CUARTETO INFERIOR

*=$C3D0             ; CODIGO DIR. INICIO
AMPM *=*+1          ; FLAG AM/PM
HOURS *=*+1         ; VALOR HORAS (INICIAL.)
MINS *=*+1          ; VALOR MINUTOS
SECS *=*+1          ; VALOR SEGUNDOS
COLOR *=*+1         ; COLOR RELOJ
DISPLY *=*+1        ; FLAG VISUAL./DESAPAR.
VECTOR *=*+2        ; ALMAC. PARA ANT. VECTOR IRQ
COUNT *=*+1         ; CONTADOR IRQ
TEMP 1 *=*+1

;
; INSERTAR CUÑA
;

   LDA  PALNTS      ; PAL O NSTC
   BEQ  NTSC        ; BIFURCACION PARA NSTC
   LDA  #HZ50       ; DEBE SER PAL
   ORA  D2CRA
   STA  D2CRA       ; ESTABLECE TODIN PARA 50 HZ
   BMI  PALDUN
NTSC
   LDA  #HZ60       ; NTSC
   AND  D2CRA
   STA  D2CRA       ; ESTABLECE TODIN PARA 60 HZ
PALDUN
   LDA  #WRITE
   AND  D2CRB       ; ESTABLECE RELOJ SIN ALARMA
   STA  D2CRB
   LDA  AMPM
   AND  #128        ; HACE UN VALOR AMPM VALIDO
   STA  AMPM
   LDA  HOURS       ; TOMA LA HORA
   JSR  BINBCD      ; CONVERSION EN BCD
   ORA  AMPM        ; OR CON FLAG AM/PM
   STA  CLOCK+3     ; ALMACENA EN RELOJ
   LDA  MINS        ; TOMA LOS MINUTOS
   JSR  BINBCO      ; CONVERSION EN BCD
   STA  CLOCK+2     ; ALMACENA EN RELOJ
   LDA  SECS        ; TOMA SEGUNDOS
   JSR  BINBCD      ; CONVERSION EN BCD
   STA  CLOCK+1     ; ALMACENA EN RELOJ
   LDA  #00         ; PONE SIEMPRE LAS DECIMAS A 0
   STA  CLOCK       ; INICIA EL RELOJ
;
   SEI              ; DESACTIVA INTERR.
   LDA  IRQVEC
   STA  VECTOR      ; GUARDA ANTIGUO VECTOR IRQ
   LDA  IRQVEC+1
   STA  VECTOR+1
;
   LDA  #<WEDGE
   STA  IRQVEC
   LDA  #>WEDGE     ; INSERTA CUÑA
   STA  IRQVEC+1
   CLI              ; ACTIVA INTERR.
   RTS
;
; QUITAR CUÑA
;
   SE I             ; DESACTIVA INTERRUPCIONES
   LDA  VECTOR
   STA  IRQVEC      ; RESTAURA VECTOR RAM
   LDA  VECTOR+1
   STA  IRQVEC+1
   CL I             ; ACTIVA INTERRUPCIONES
   RTS

;
; AQUI COMIENZA LA CUÑA
;
WEDGE
   LDA  COUNT       ;
   CMP  #RATE       ; RELOJ CON ESTA IRQ?
   BEQ  CONT        ;
OUT
   JMP  EXIT        ; NO
CONT
   LDA  #$FF        ; RESTABLECE CONTADOR IRQ
   STA  COUNT
   LDA  DISPLY      ; VISUALIZACION?
   BEQ  OUT         ; NO... BIFURCAR

   LDA  CLOCK+3     ; TOMA HORAS/AM/PM
   TAX              ; PONE UNA COPIA EN EL REG X
   AND  #$80        ; TOMA AM/PM
   BNE  PM          ; BIFURCA SI ES PM
   LDA  #AY         ; VISUALIZA 'A'
   JMP  MERIDP
PM
   LDA  #PEE        ; VISUALIZA 'P'
MERIDP
   STA  SCNLOC+10
   LDA  COLOR       ; TOMA COLOR
   STA  COLLOC+10   ; ESTABLECE EL COLOR
   LDA  #EM         ; VISUALIZA 'M'
   STA  SCNLOC+11
   LDA  COLOR       ; ESTABLECE COLOR
   STA  COLLOC+11
;
; PONE LAS HORAS
;
   TXA              ; TOMA LA HORA
   AND  #$10        ; SE DESEA EL DIGITO SUP.
   JSR  HIDIGT      ; TOMA CODIGO PANT.
   STA  SCNLOC      ; LO VISUALIZA
   LDA  COLOR
   STA  COLLOC      ; ESTABLECE COLOR
   TXA              ; TOMA EL BYTE DE NUEVO
   JSR  LODIGT      ; TOMA EL DIGITO INF.
   STA  SCNLOC+1    ; LO VISUALIZA
   LDA  COLOR       ;
   STA  COLLOC+1    ; ESTABLECE COLOR
;
   LDA  #COLON      ; SEPARADOR HRS/MINS
   STA  SCNLOC+2
   LDA  COLOR
   STA  COLLOC+2
;
; PONE LOS MINUTOS
;
   LDA  CLOCK+2     ; TOMA MINUTOS
   TAX
   JSR  HIDIGT      ; HACE DIGITO SUP.
   STA  SCNLOC+3    ; LO VISUALIZA
   LDA  COLOR
   STA  COLLOC+3    ; Y COLOREA
   TXA              ; TOMA EL BYTE DE NUEVO
   JSR  LODIGT      ; HACE EL DIGITO INF.
   STA  SCNLOC+4    ; LO VISUALIZA
   LDA  COLOR       ;
   STA  COLLOC+4      ESTABLECE COLOR
;
   LDA  #SLASH      ; SEPARADOR MIN./SEG.
   STA  SCNLOC+5
   LDA  COLOR
   STA  COLLOC+5
;
; PONE LOS SEGUNDOS
;
   LDA  CLOCK+1     ; TOMA SEGUNDOS
   TAX
   JSR  HIDIGT      ; HACE DIGITO SUP.
   STA  SCNLOC+6    ; LO VISUALIZA
   LDA  COLOR
   STA  COLLOC+6    ; Y COLOREA
   TXA              ; TOMA EL BYTE DE NUEVO
   JSR  LODIGT      ; HACE EL DIGITO INF.
   STA  SCNLOC+7    ; LO VISUALIZA
   LDA  COLOR
   STA  COLLOC+7    ; Y COLOREA
;
   LDA  #POINT      ; SEPARADOR SEG./DECIMAS
   STA  SCNLOC+8
   LDA  COLOR
   STA  COLLOC+8
;
; PONE LAS DECIMAS
;
   LDA  CLOCK       ; TOMA EL VALOR DE DECIMAS
   ADC  #DIGIT      ; SUMA $30 PARA COD. PANT.
   STA  SCNLOC+9    ; LO VISUALIZA
   LDA  COLOR
   STA  COLLOC+9    ; Y COLOREA
;
;
EXIT
   INC  COUNT       ; INCREMENTA CONT. IRQ
   JMP  (VECTOR)    ; VA AL RESTO DE IRQ
;
; SUBRUTINAS
;
HIDIGT
   LSR  A
   LSR  A           ; MUEVE CUART. SUP. AL INF.
   LSR  A
   LSR  A
   CLC
   ADC  #DIGIT      ; SUMA $30 PARA COD. PANT.
   RTS

LODIGT
   AND  #TRNCLO     ; DESENMASCARA CUART. SUP.
   CLC
   ADC  #DIGIT      ; SUMA $30 PARA COD. PANT.
   RTS

;
; CONVERSION DE BINARIO A BCD
;
BINBCD
   LDY  #$FF
   SEC
D10
   INY
   SBC  #10         ; RESTA 10 HASTA -VE
   BCS  D10
   ADC  #10         ; SUMA DE 10 PARA ABAJO
   STA  TEMP 1      ; ALMACENA EL RESTO
   TYA              ; TOMA NUM PF 10S RESTADO
   ASL  A
   ASL  A
   ASL  A           ; LO DESPLAZA AL CUART. SUP.
   ASL  A
   ORA  TEMP 1      ; PONE RESTO EN CUART. INF.
   RTS
```

Impreso con el amable permiso
de los autores y de Ellis Horwood LTD.
Tomado de *Mastering the Commodore 64*,
por Jones y Carpenter

Editorial Delta, S.A.



10081

9 788485 822836